НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Веб-додаток для автоматизації офлайн-перевірки письмових тестів»

Виконала:
студентка IV курсу, групи КП-51
Чумак Карина Сергіївна _____

Керівник:
Ст. викл. кафедри ПЗКС,
Гадиняк Р.А. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
Доцент кафедри ММСА ІПСА, к.т.н.,
Дідковська М.В. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____ 2018 р.

ЗАВДАННЯ

на дипломний проект студенту

Чумак Карині Сергіївні

1. Тема проекту «Веб-додаток для автоматизації офлайн-перевірки письмових тестів», керівник проекту Гадиняк Руслан Анатолійович, старший викладач, затверджені наказом по університету від «22» травня 2019 року № 1331-С.

2. Термін подання студентом проекту «20» червня 2019 р.

3. Вихідні дані для дипломного проектування: див. Технічне завдання.

4. Зміст проекту:
   - провести аналіз існуючих аналогів та аналіз предметної області;
   - провести аналіз наукових робіт з розпізнавання та сегментації рукописного тексту;
   - провести збір даних для тестування роботи системи;
   - провести аналіз існуючих наборів даних для створення моделі розпізнавання рукописного тексту;
   - сформувати вимоги до користувацького інтерфейсу програми;
   - реалізація алгоритму сегментації рукописного тексту;
   - реалізація алгоритму розпізнавання рукописного тексту;
   - розробити модель бази даних для збереження тестів;
   - виконати програмну реалізацію web-додатку відповідно до вимог технічного завдання;
   - провести тестування роботи програми.

5. Перелік обов'язкового ілюстративного матеріалу:
   - структура бази даних (креслення);
   - алгоритм трансформації по чотирьох точках (креслення);
   - архітектура нейронної мережі (плакат);

– структурна схема системи (плакат).

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент кафедри ПЗКС, к.т.н.		

7. Дата видачі завдання «31» жовтня 2018 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	16.11.2018	
2.	Розробка та узгодження технічного завдання	09.12.2018	
3.	Підготовка матеріалів першого розділу дипломного проекту	30.12.2018	
4.	Розроблення структури web-додатку	16.01.2019	
5.	Підготовка матеріалів другого розділу дипломного проекту	01.03.2019	
6.	Розроблення алгоритму розпізнавання рукописного тексту	22.03.2019	
7.	Програмна реалізація і тестування web-додатку	12.04.2019	
8.	Підготовка третього розділу дипломного проекту	25.04.2019	
9.	Підготовка четвертого розділу дипломного проекту	02.05.2019	
10.	Підготовка графічної частини дипломного проекту	19.05.2019	
11.	Оформлення документації дипломного проекту	26.05.2019	
12.	Вивчення літератури за тематикою проекту	16.11.2018	

Студент _____ Чумак К.С.

Керівник проекту _____ Гадиняк Р.А.

АНОТАЦІЯ

Дана робота присвячена розробленню web-сервісу для автоматизації офлайн-перевірки письмових тестів.

У роботі виконано порівняльний аналіз існуючих рішень для перевірки тестів з зображення письмової роботи студента, проаналізовано методи розпізнавання письмового тексту та перевірки відповідей, обґрунтовано вибір технологій та допоміжних бібліотек серверної та клієнтської частин для реалізації даного web-сервісу. Розроблений web-сервіс надає викладачам можливість перевіряти написані від руки тестові завдання, сфотографувавши їх на мобільний телефон, а також створювати та редагувати свої тести, та моніторити статистику успішності учнів. Процес розпізнавання тексту здійснюється автоматично за допомогою засобів машинного навчання. Система надає гнучкість у створенні тестів, а саме методу їх оцінки та типу завдання. Результатом роботи системи є рекомендаційні настанови для питань з відкритою відповіддю, а також повна оцінка для питань закритого типу. Усі дані про проходження кожним студентом тесту зберігаються у базі даних, та доступні для порівняння та аналізу.

У даному дипломному проекті розроблено: архітектуру серверної та клієнтської частини web-сервісу, алгоритм попередньої обробки зображення, алгоритм трансформування сторінки, алгоритм сегментації сторінки на рядки та слова, алгоритм розпізнавання слів з зображення, алгоритм формування масиву відповідей, алгоритм виставлення оцінки, а також графічні елементи та дизайн web-сторінок.

ABSTRACT

This diploma is devoted to the development of a web-service for automating the offline grading of written tests.

In this work a comparative analysis of existing solutions of checking the tests based on the image of the student's written work was made, the methods for recognizing the written text and checking the answers were analyzed, the choice of technologies and auxiliary libraries of the server and client parts for the implementation of this web-service is grounded. The developed web-service provides teachers with the opportunity to test handwritten test tasks by photographing them on a mobile phone, as well as create and edit their own tests, and monitor students progress statistics. The process of recognizing the text is automatically done using machine learning tools. The system provides flexibility in the development of tests, namely, the method of their evaluation and the type of task. The result of the systems work is the guidance guidelines for open-ended questions, as well as a full evaluation for closed-ended issues. All student test data is stored in a database and is available for comparison and analysis.

This diploma project contains the following: the architecture of the server and client part of the web-service, the algorithm of preliminary processing of the image, the algorithm of transforming the page, the algorithm of segmentation of the page into lines and words, the algorithm of word recognition from the image, the algorithm of forming an array of responses, the algorithm of evaluation, as well as graphic elements and design of web pages.

ДП.045440-01-90 Веб-додаток для автоматизації офлайн-перевірки письмових тестів. Відомість проекту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проекту		
ДП.045440-02-91	Веб-додаток для	5	
	автоматизації офлайн-		
	перевірки письмових		
	тестів. Технічне		
	завдання		
ДП.045440-03-81	Веб-додаток для	70	
	автоматизації офлайн-		
	перевірки письмових		
	тестів. Пояснювальна		
	записка		
ДП.045440-04-51	Веб-додаток для	4	
	автоматизації офлайн-		
	перевірки письмових		
	тестів. Програма та		
	методика тестування		
ДП.045440-05-34	Веб-додаток для	11	
	автоматизації офлайн-		
	перевірки письмових		
	тестів. Керівництво		
	користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Веб-додаток для	1	
	автоматизації офлайн-		
	перевірки письмових		
	тестів. Структура бази		
	даних. ERD діаграма		
ДП.045440-07-99	Веб-додаток для	1	
	автоматизації офлайн-		
	перевірки письмових		
	тестів. Алгоритм		
	трансформації по		
	чотирьох точках.		
	Діаграма діяльності		
ДП.045440-08-98	Веб-додаток для	1	
	автоматизації офлайн-		
	перевірки письмових		
	тестів. Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ОФЛАЙН-ПЕРЕВІРКИ ПИСЬМОВИХ ТЕСТІВ

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А.Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ К.С.Чумак

2018

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-додаток для автоматизації офлайн-перевірки письмових тестів.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання викладачами в якості допоміжного інструменту для перевірки письмових офлайн-тестів з будь-якої дисципліни англійською мовою.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Веб-додаток повинен забезпечувати такі основні функції:

1) можливість створити новий тест у системі;
2) можливість додати питання до створеного тесту;
3) можливість внести приклад еталонної відповіді на запитання;
4) можливість зробити фото написаної студентом відповіді на папері;
5) розпізнавання написаного тексту;
6) порівняння написаної студентом відповіді з еталонною відповіддю, створеною викладачем;

7) виставлення рекомендованої оцінки;

8) можливість коригування оцінки;

9) можливість створення списку групи;

10) можливість імпорту списку групи з csv файлу;

11) можливість збереження виставлених оцінок;

12) автоматичне створення графіків оцінок.

Розробку виконати мовою Python.

Додаткові нефункціональні вимоги:

1) динамічне відображення питань при створенні тесту;

2) динамічна зміна побудованого графіку при зміні параметрів;

3) мінімалістичний дизайн сторінок інтерфейсу;

4) наявність повідомлень про помилки у випадку некоректно введених користувачем даних;

5) час обробки та розпізнавання зображення – до 20 секунд.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;

2) програма та методика тестування;

3) керівництво користувача;

4) креслення:

- Структура бази даних. ERD діаграма.

- Алгоритм трансформування по чотирьох точках. Діаграма діяльності.

6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення літератури за тематикою роботи……..……..30.11.2018

Розроблення та узгодження технічного завдання…..……...15.12.2018

Розроблення структури веб-додатку...…….…….…….....03.01.2019

Розроблення алгоритму розпізнавання……..…....10.01.2019

Програмна реалізація веб-додатку…….....10.02.2019

Створення інтерфейсу...…….....10.03.2019

Тестування веб-додатку...…….....03.04.2019

Підготовка матеріалів текстової частини проекту…….18.04.2019

Підготовка матеріалів графічної частини проекту……..…12.05.2019

Оформлення технічної документації проекту…..……..…25.05.2019

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А.Дичка

"___" _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ОФЛАЙН-ПЕРЕВІРКИ ПИСЬМОВИХ ТЕСТІВ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А. Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ К.С. Чумак

2019

ЗМІСТ

ВСТУП

В процесі навчання у будь-якій сфері діяльності незмінною складовою є проведення тестування та моніторингу знань учнів або студентів. Дуже часто викладачам доводиться перевіряти велику кількість типових тестів, витрачаючи на це багато часу. Для вирішення цієї проблеми створюються системи онлайн тестувань, проте й досі найпопулярнішим видом тестування є звичайні рукописні офлайн тести на папері. Це пов’язано з тим, що для проведення онлайн тесту необхідно забезпечити студентів доступом до комп’ютерів, електроенергією та якісним інтернет – з’єднанням. Офлайн-тестування залишається найбільш репрезентативним видом перевірки знань, що призводить до того, що викладачам часто доводиться робити багато монотонної однотипової роботи під час перевірки. Деякі викладачі друкують бланки для тестів, щоб якось структурувати відповіді студентів та зекономити час на перевірці робіт.

З огляду на це, створення системи, яка може автоматизувати перевірку офлайн-тестів є актуальною задачею.

Дана робота присвячена розробці веб-додатку для спрощення процесу перевірки тестів з використанням технологій оптичного розпізнавання символів.

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

OCR (англ. Optical character recognition, укр. оптичне розпізнавання символів) – переведення зображення, що містить будь-який текст, у текстові дані.

PDF (англ. Portable Document Format) – формат файлу, який не залежить від пристроїв.

CSV (англ. Comma separated value) – формат запису даних, при якому атрибути записуються через кому.

ОС – операційна система.

Machine Learning (ML, укр. машинне навчання) – набір алгоритмів та статистичних моделей, які використовуються в комп'ютерних системах для ефективного виконання певних типів завдань без використання явних інструкцій, натомість покладаючись на виявлення зв'язків та шаблонів у даних.

GIL (Global Interpreter Lock) – це механізм інтерпретатора, який забезпечує виконання лише одного потоку в межах одного процесу за один квант часу.

IoT (Internet of things) – концепція обчислювальної мережі фізичних пристроїв, які оснащені вбудованими технологіями для взаємодії один з одним та з зовнішнім середовищем.

ORM (англ. Object-Relational Mapping, укр. об'єктно-реляційне співставлення) – технологія програмування, що пов'язує бази даних з концепціями об'єктно-орієнтовних мов програмування.

API (англ. application programming interface, укр. програмний інтерфейс додатку) – набір засобів (класів, функцій, методів, процедур), завдяки яким одна комп'ютерна програма може взаємодіяти з іншою.

Датасет (англ. Dataset) – набір даних, найчастіше відповідає змісту таблиці бази даних, або статистичній матриці даних тощо.

Deslanting (від англ. Slant – нахил) – процес позбавлення нахилу рукописних літер, переведення їх до однакового нахилу для будь-якого почерку з метою покращення результатів розпізнавання.

Skew correction (з англ. корекція перекосу) – процес вирівнювання рядків рукописного тексту до паралельних прямих з метою покращення сегментації сторінки.

Natural Language Processing, NLP (укр. Обробка природньої мови) – підрозділ машинного навчання, який вивчає проблеми комп'ютерного аналізу та синтезу природніх мов.

Character Error Rate, CER (укр. частка помилкових символів) – метрика, що використовуються при розробці алгоритмів розпізнавання текстових даних. Визначається як відношення помилково визначених символів до загальної кількості симолів у реченні.

Word Error Rate,WER (укр. частка помилкових слів) – метрика, що визначається як відношення помилково визначених слів до загальної кількості слів у реченні.

Згорткова нейронна мережа (англ. convolutional neural network, CNN) – спеціальна архітектура штучних нейронних мереж, що націлена на ефективне розпізнавання образів та входить в склад технологій глибокого навчання.

Рекурентна нейронна мережа (англ. Recurrent neural network, RNN) – вид нейронних мереж, в яких зв'язки між елементами утворюють спрямовану послідовність, що імітує внутрішню пам'ять нейронів головного мозку.

LSTM (англ. Long short-term memory, укр. Довга короткочасна пам'ять) – різновид архітектури рекурентної нейронної мережі.

CTC (англ. Connectionist temporal classification, укр. коннекційна часова класифікація) – це функція, що дозволяє рекурентним нейронним мережам навчатися для розпізнавання послідовності слів без початкового вирівнювання вхідних та вихідних послідовностей.

Pooling layer (укр. агрегувальний шар) – частина згорткової нейронной мережі, яка відповідає за нелінійне зменшення розмірності.

Tf-idf (англ. TF – term frequency, IDF – inverse document frequency) – статистична міра, що використовується для оцінки важливості слова в контексті документа, який є частиною колекції документів.

Краудсорсинг (англ. crowd – толпа, source – ресурс) – залучення до вирішення проблем інноваційної діяльності широкого кола людей для спрощення та пришвидшення процесу.

Amazon Web Services (AWS) – комерційна публічна платформа для хмарних обчислень.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз процесу тестування знань

Тестування, або ж іспит, – це завдання, покликане виміряти рівень знань, навичок, здібностей, фізичної підготовки або кваліфікації випробуваного. Тестування може проводитись усно, на папері, за допомогою електронних пристроїв, і може відрізнятись за стилем, вимогами та строгістю: використання допоміжних матеріалів може бути заборонене, або дозволене використання одного або кількох додаткових інструментів, наприклад, довідника або калькулятора, для надання відповідей на тестові питання [1].

Оцінка студента може бути нормалізована після проведення тестування. Норма встановлюється незалежно, або шляхом статистичного аналізу великої кількості учасників. Для забезпечення правової захищеності випробуваних вводяться стандартизовані тести – це будь-які тести, які проводяться та оцінюються послідовно, не упереджено та за певними встановленими правилами.

Тести можна класифікувати багатьма способами. Основна класифікація – за спрямованістю тесту: оцінювати можна знання людини, його здатності, особисті якості, інтелект, окремі психічні функції (увага, пам'ять, уява) тощо.

Традиційний тест має цілісність і структуру. Він містить завдання, вимоги до відповідей, критерії оцінювання кожного завдання і рекомендації по інтерпретації тестових результатів. Результат традиційного тесту залежить від кількості питань, на які було дано правильну відповідь.

Одними з нетрадиційних тестів є інтегративні та адаптивні тести [2]. Інтегративними називаються тести, що складаються з системи завдань, спрямованих на узагальнену перевірку підготовленості випробуваного. Такі тести містять завдання, як потребують знань

декількох різних дисциплін, а також вміння їх пов’язувати між собою та аналізувати.

Адаптивні тести представляють собою автоматизовану систему тестування, в якій кожне завдання характеризується такими параметрами, як рівень складності та здатність диференціювати учнів. Ці системи створені у вигляді банку завдань, упорядкованих відповідно до характеристик завдань. Таким чином, тести змінюються та підлаштовуються під кожного учня індивідуально.

Розроблюване програмне забезпечення покликане спростити процес перевірки тестів викладачам вищих навчальних закладів, в яких здебільшого використовуються традиційні тести. Для перевірки адаптивних тестів необхідно реалізовувати додаткову бізнес-логіку.

Усі традиційні тести можна поділити на дві категорії: відкритого типу (коли учень має сам вписати відповідь) та закритого (коли учень має обрати одну із запропонованих відповідей).

Завдання закритого типу можуть мати наступну форму:

1) обрати одну відповідь із декількох;
2) обрати декілька відповідей із декількох;
3) встановити відповідність;
4) визначити послідовність (порядок написаних літер має значення).

Перевірка завдань з закритою відповіддю є монотонною роботою, під час якої викладач може легко помилитися та яку можна автоматизувати. Такі завдання, як правило, є однозначними та під час їх перевірки не може виникати спірних питань.

Відкриті завдання можуть вимагати від учня розгорнутої відповіді, опису процесу, послідовності подій, переклад тексту, розкриття дужок, виведення формули, побудови схеми або ж просто доповнення контексту одним-двома словами.

В залежності від дисципліни, з якої проводиться тестування, відкрита відповідь на тест може також містити один, або декілька наступних елементів:

1) графік;

2) схема;

3) формула;

4) програмний код;

5) малюнок;

6) будь-які специфічні для цієї сфери символи.

Процес перевірки відкритих завдань є досить складним, оскільки часто вимагає від викладача розуміння ходу думок студента. З точки зору розроблення алгоритму для автоматичної перевірки відповідей, можна виділити наступні перешкоди, які несуть відкриті відповіді: математичну формулу можна написати декількома способами, або з різними позначеннями; відповідь, яку розраховує побачити викладач, може бути не єдиною правильною; графік певною мірою залежить від художніх здібностей студента та його дуже важко співставити з еталонним графіком, якого може не існувати.

Отже, на відміну від закритих відповідей, перевірку відкритих набагато важче автоматизувати і найкращий варіант – перевіряти загальну тему відповіді та кількість слів, що співпадають з еталонною відповіддю, внесеною викладачем заздалегідь, а потім рекомендувати викладачеві оцінку, залишаючи остаточне рішення за ним.

1.2. Аналіз існуючих програмних рішень

Перед початком розроблення необхідно провести аналіз програмних рішень, які також покликані спростити процес проведення тестування. Можна зазначити, що існує дуже багато систем онлайн тестування, проте досить мало програм, які базуються саме на розпізнаванні написаної на листі відповіді.

Найбільш популярними та успішними з них є ZipGrade, GradeCam та Essay Grader.

1.2.1. *ZipGrade*

Мобільний додаток ZipGrade – система для перевірки тестів, покликана повністю автоматизувати перевірку тестів. Надає наступні функціональні можливості:

1) створення нового тесту;
2) внесення правильних відповідей;
3) перегляд аналітики по тесту;
4) експорт звіту про тест у форматі PDF та CSV;
5) використання з декількох пристроїв.

При цьому безкоштовна версія накладає наступні обмеження на користувачів:

1) перевірка 100 робіт в місяць;
2) фіксована кількість запитань у тесті.

Крім зазначених вище безкоштовних функцій, користувач також може сплатити та отримати наступні додаткові можливості:

1) перевірка тесту без вказання імені студента;
2) офлайн перевірка без доступу до Інтернет;
3) імпорт списків учнів із CSV файлу;
4) створення тестів з будь-якою кількістю запитань.

Недоліками цієї системи можна вважати:

1) недостатня гнучкість у створенні тестів;
2) відсутність можливості вказання кількості балів за певне питання;
3) необхідність друкування бланків відповідей для усієї групи;
4) неможливість створення закритих тестів;
5) неможливість створення відкриті тести з декількома відповідями;

6) неможливість створення тестів на співставлення, або визначення послідовності;
7) обмеження кількості варіантів тесту – не більше 5.

Отже, цей мобільний додаток дійсно здатен робити перевірку тестів повністю автоматичною з моменту створення тесту і завантаження фотографії. Після перевірки оцінка виставляється миттєво, оскільки усі тести мають лише одну правильну відповідь та оцінка може вважатися однозначною. Користувачі цієї системи не мають змоги створювати та формувати інші типи питань, окрім закритих тестів з однією відповіддю, що і є основним недоліком цього додатку. Проте декілька функцій є дуже корисними і запозичені для цього дипломного проекту.

1.2.2. *GradeCam*

GradeCam – це веб-додаток, який має більш широкі функціональні можливості, ніж попередній продукт. Його принцип дії схожий на ZipGrade, проте він має більш привабливий інтерфейс аналітики, а також дає можливість розпізнавання тексту. Ця система була створена для автоматизації процесу викладання від створення тестів, збереження їх у базі контрольних робіт, до виставлення оцінки, моніторингу успішності учнів та генерації звітності для керівництва, батьків, студентів тощо.

Перевагами GradeCam можна вважати наступні можливості:

1) створення тестів на комп’ютері зручніше, ніж на мобільному додатку;
2) адаптування форми відповіді під різні дисципліни;
3) сканування роботи будь-якою камерою: на мобільному пристрої, на ноутбуці та комп’ютері, на зовнішньому пристрій;
4) створення тестів з декількома відповідями;
5) існує такий тип питання, який не перевіряється системою, а лише зберігається його зображення, щоб викладач міг

перевірити його вручну пізніше, при цьому не треба було б нести роботи для перевірки додому;

6) розпізнавання рукописного тексу відкритих відповідей;
7) збереження усіх оцінок та робіт;
8) збереження інформації про правильні відповіді по кожному студенту та кожному тесту;
9) автоматична побудова графіків з багатьма параметрами;
10) інтеграція з іншими додатками виставлення оцінок;
11) формування звітності з успішності груп та учнів;
12) сканування вже перевіреної вручну роботи для збереження цієї оцінки у системі;
13) автоматичне виставлення результуючої оцінки на основі оцінок за тести;
14) студентський портал, в якому студенти мають змогу переглядати свої поточні оцінки;
15) повідомлення студентів про щойно виставлену оцінку;
16) перегляд студентами перевіреної роботи.

Недоліками системи є:

1) працює лише на створених у системі формах;
2) необхідність друкувати велику кількість форм для відповідей;
3) питання з відкритими відповідями також мають бути внесені у форму друкованими літерами, кожна літера у окремій комірці на формі;
4) не підтримує інші мови, окрім англійської;
5) високі ціни за продукт, які варіюються в залежності від кількості учнів або від кількості перевірених робіт.

Система має велику кількість дуже корисних функцій, проте викладачі все одно мусять друкувати спеціальні форми відповідей, що несе додаткові витрати часу та грошей, а також унеможливлює проведення тестів без підготовки, наприклад, якщо залишився вільний

час на занятті. Якщо університет, або інший навчальний заклад не співпрацює з цією системою, окремому викладачу її використання буде дорого коштувати.

1.2.3. *Essay Grader*

Essay Grader – додаток, покликаний полегшити перевірку творів, тобто саме повноцінного рукописного тексту, а не лише закритих тестів. Ця система дає змогу викладачам завантажити зображення твору, перевірити його на наявність плагіату, знайти граматичні помилки та надати викладачеві інтерфейс для цифрового коригування речень, а також можливості писати різні коментарі та одразу ж надсилати результат учням. В цій системі є розпізнавання рукописного тексту, проте вона не здатна автоматизувати перевірку тестів, а спрямована виключно на допомогу в перевірці творів. Цей додаток не дає можливості ведення звітності, збереження студентських оцінок та перегляду статистики. Отже, ця система хоч і має схожу тематику, не є аналогом до розроблюваного дипломного проекту.

1.3. Висновок та аналіз вимог до функціональності

В результаті дослідження сфери та аналізу існуючих рішень, можна стверджувати, що проблема швидкої перевірки тестів є актуальною, проте спроби її вирішити зводяться до додатків з жорсткою логікою, які не дають користувачеві змогу підлаштовувати їх під себе, а саме створювати тести відкритого типу, тести на встановлення відповідності, або на визначення послідовності та змінювати кількість балів за певне запитання тесту. Ті системи, що дають можливість гнучкого створення тестів, все одно накладають багато обмежень на способи запису відповідей.

В процесі аналізу вимог було сформовано наступні функціональні вимоги:

1. Авторизація у системі для роботи з нею. Для неавторизованих користувачів функції системи недоступні. Єдина роль користувача – викладач.
2. Можливість залишатись авторизованим у системі після закриття браузера.
3. Створення тесту. Викладач має змогу створити новий тест, вказавши назву дисципліни, до якої він відноситься, а також опис тесту, після цього послідовно додати до нього питання, вказуючи їх тип, текст запитання, правильну відповідь та кількість балів.
4. Розпізнавання відповідей. За допомогою веб-інтерфейсу завантажити фотографію, виокремити з неї написаний текст та надрукувати його в спеціальному полі для демонстрації порівняння з відповіддю.
5. Перевірка відповідей. Ті відповіді, що повністю співпадають з еталонною, оцінити, а ті, що не співпадають,– залишити користувачеві на додаткову перевірку.
6. Виставлення оцінки. Перевірка типу тестів, якщо усі з них були закритими та однозначними – виставити оцінку в таблицю, інакше – підрахувати рекомендовану оцінку та надати можливість викладачу її скоригувати.
7. Збереження результатів. Усі оцінки зберігаються у базі даних для подальшого аналізу.
8. Імпорт списку групи з CSV. При створенні нової групи користувач завантажує файл CSV зі списком, після чого вона одразу показується у списку груп викладача.
9. Перегляд графіків успішності. Користувач має змогу переглянути графіки, побудовані на основі оцінок учнів обраної

групи за обраний тест. Доступно два типи графіків: гістограма та boxplot. При обранні типу графіка boxplot та усіх груп, boxplot має бути побудований для кожної групи окремо на одному графіку.

Додаткові нефункціональні вимоги:

1. Динамічне завантаження списку питань. На сторінці створення тесту під час додавання нового питання воно одразу додається у список та показується на сторінці.
2. Динамічна зміна графіків. При зміні параметрів побудови графіків, вони будуються динамічно без перезавантаження сторінки.
3. Завантажена робота може займати не усю фотографію.
4. Фото може бути зроблене під кутом до площини роботи.

2. АНАЛІЗ МОВ ПРОГРАМУВАННЯ ТА ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ WEB-ДОДАТКІВ

2.1. Обгрунтування вибору web-додатку в якості типу програмного забезпечення

Розроблювана система має використовувати засоби машинного навчання для розпізнавання рукописного тексту. Існують різні алгоритми машинного навчання, але ті, які здатні впоратись із завданнями Computer vision, зазвичай мають декілька мільйонів параметрів та є досить важкими з точки зору обчислень та пам’яті, що займає програма. Найбільш оптимальна архітектура, яка здатна впоратися із поставленою задачею, – це клієнт-сервер. Для того, щоб уникнути необхідності стискання моделі та зменшення кількості параметрів, розпізнавання (inference) виконується на стороні серверу, а користувачу надається інтерфейс для надсилання даних на сервер з клієнту.

В якості інтерфейсу можна використати мобільний застосунок, або ж веб-додаток. Проаналізуємо їх переваги та недоліки з точки зору системи, що розроблюється.

Переваги мобільного застосунку:

1) простіший у використанні;
2) більша продуктивність завдяки взаємодії з ОС;
3) За статистикою, користувачі мобільних телефонів більше часу проводять у мобільних додатках, ніж в Інтернеті [3].

Недоліки мобільного застосунку:

1) вища вартість розроблення;
2) необхідність розроблювати декілька додатків для різних операційних систем;
3) неможливість застосувати будь-яку камеру, окрім камери мобільного телефону;
4) потребує встановлення та займає пам’ять телефону;

5) важче підтримувати.

Переваги веб-застосунку:

1) не потребують завантаження та встановлення;
2) можливість використовувати на будь-якому пристрої;
3) легші у розробці та підтримці;
4) можливість використовувати будь-яку зовнішню камеру, або сканер для завантаження фото.

Недоліки веб-застосунку:

1) повільніший, ніж мобільний додаток.

Отже, з огляду на описані переваги та недоліки обох типів клієнтських додатків, було обрано веб-додаток в якості типу програмного забезпечення. Ним можна користуватись з будь-якого типу пристроїв, і з мобільних у тому числі, проте не потрібно виконувати розроблення для різних операційних систем, що істотно спрощує створення даної системи. Крім цього, початок роботи із веб-застосунком набагато простіший, ніж з мобільним додатком, оскільки користувачі мають лише зареєструватися у системі та можуть одразу його використовувати. Для цього не потрібно завантажувати додаток та чекати, поки він встановиться.

2.2. Мова програмування

Найпоширенішими мовами програмування в сфері Machine Learning є Python та C/C++, в той час, як для програмування веб-додатків найбільш популярними є Python та Javascript.

2.2.1. *Python*

Python вважається найбільш популярною мовою серед усіх мов розвитку AI завдяки його простоті. Його синтаксис дуже простий та легко піддається вивченню. Це дозволяє швидко та ефективно реалізовувати алгоритми AI та будувати прототипи нових моделей. В

порівнянні з іншими мовами, такими як Java, C ++ або Ruby, час розроблення мовою Python значно менший, що є великою перевагою у науковій сфері, де необхідно багато експериментувати та якомога швидше отримувати результати. Завдяки цьому розвиток машинного навчання безпосередньо пов’язаний з мовою Python. Існує дуже велика кількість бібліотек та фреймворків для цієї мови, які полегшують процес маніпулювання даними, їх обробку, аналіз, збереження тощо.

Python підтримує об'єктно-орієнтовані, функціональні та процедурно-орієнтовані стилі програмування.

До переваг мови Python можна віднести:

1) динамічна типізація;
2) підтримка модульності;
3) інтерпретована мова;
4) підтримка об’єктно-орієнтованого програмування;
5) інтеграція з C/C++, якщо можливостей Python недостатньо;
6) автоматична збірка сміття;
7) відсутність протікання пам’яті;
8) простота та лаконічність;
9) безліч бібліотек та фреймворків, особливо для машинного навчання;
10) кросплатформність;
11) багато прикладів коду для навчання;
12) підходить для веб-розроблення.

Деякі з вищезазначених переваг є одночасно і недоліками. Наприклад, простота мови досягається мінімізацією участі розробника у керуванні пам’яттю, а динамічна типізація також накладає певні обмеження у вигляді необхідності більш ретельного тестування та виправлення помилок.

Python сильний у desktop і серверних платформах, але слабкий у мобільних платформах. Існує дуже мало мобільних додатків,

розроблених з використанням Python, і мова рідко зустрічається на клієнтській стороні веб-додатків.

Недоліками мови Python є:

1) швидкість роботи коду;
2) використання пам’яті;
3) багатопотоковість в Python може мати непередбачувані результати.

Повільне виконання програм, написаних мовою Python, пов’язане з декількома причинами. По-перше, це мова з динамічною типізацією. Це дуже зручно з точки зору розробника, оскільки йому не треба витрачати час на явне визначення типів даних. Проте всю цю роботу доводиться виконувати інтерпретатору, що збільшує час виконання програм [4, 5].

GIL – Global Interpreter Lock – це механізм інтерпретатора, який забезпечує виконання лише одного потоку в межах одного процесу за один квант часу. Оскільки інтерпретатор Python використовує GIL, використання паралельних обчислень стає складним завдання для розробника, якому необхідно для цього створювати декілька процесів та реалізовувати механізми міжпроцесної взаємодії, що також впливає на час виконання.

Отже, мова Python здатна впоратися з багатьма різними завданнями, як у веб-розробці, так і у розробці алгоритмів машинного навчання.

2.2.2. *C/C++*

C/C++ – дуже поширені мови програмування в сферах розроблення апаратного забезпечення, оскільки код, написаний на C/C++, займає суттєво менше місця та більш підходить для реалізації алгоритмів у сфері IoT. Для цих мов існує багато різних бібліотек з широкою підтримкою. Оскільки це компільовані мови, програми, написані на них, потребують набагато менше часу на виконання, ніж Python скрипти.

Переваги мов C/C++:

1) масштабовність;
2) можливість роботи на низкому рівні з пам’яттю, адресами, портами;
3) швидкодія;
4) можливість створення узагальнених алгоритмів для різних типів даних, їхня спеціалізація, і обчислення на етапі компіляції, з використанням шаблонів;
5) підтримуються різні стилі та технології програмування, включаючи традиційне директивне програмування, ООП, узагальнене програмування, метапрограмування (шаблони, макроси).

Стандарти мов C/C++ дозволяють розробникам робити багато таких речей, що можуть викликати несподівану поведінку програми та спричинити проблеми з безпекою. C++ не керує пам’яттю програми, змушуючи програміста робити це самостійно.

Недоліками мов C/C++ є:

1) погана підтримка модульності;
2) складність синтаксису;
3) складність у роботі з пам’яттю: можливе протікання;
4) бібліотеки з машинного навчання для C/C++ потребують високого рівня підготовки розробника;
5) не підходить для веб-програмування;

Мови C/C++ активно використовуються у продуктах, пов’язаних з CV, коли виникає необхідність обробки великої кількості зображень, або відео матеріалів швидко. Одним з можливих способів використання є розроблення мовою Python основного додатку, деяких частин алгоритмів, інтерфейсу тощо, та розроблення модулів обробки зображень мовами C/C++ [6].

Оскільки завдання полягає у створенні веб-додатку з алгоритмом машинного навчання, для розроблення було обрано мову Python. Вона добре підходить для створення веб-серверу, для опрацювання зображень та підготовки їх до подальшого аналізу та для реалізації нейронної мережі.

2.3. Порівняння бібліотек для веб-розроблення мовою Python

Існує багато фреймворків, які надають можливість розробнику створювати веб-додатки, використовуючи Python як мову програмування на стороні сервера. Два з найбільш широко використовуваних – Flask та Django. Хоча обидва ці фреймворки дуже популярні, вони абсолютно різні з точки зору того, що в них вже реалізовано, і що має реалізувати розробник. Flask представляє собою мікрофреймворк, який орієнтований на простоту та мінімалізм . Він реалізує мінімальні функції, залишаючи розробнику повну свободу вибору модулів та надбудов. З іншого боку, Django має більш широкий підхід, надає адміністративну панель та ORM прямо «з коробки».

Django був створений для швидкого розвитку складних веб-додатків. Розробники мають всі інструменти, необхідні їм для реалізації, і розробляють легко масштабовані, надійні та підтримувані веб-програми у рекордний час. Простота Flask також дозволяє досвідченим розробникам створювати менші програми за короткі терміни.

Одна з найбільших сильних сторін Flask – його мінімалізм і простота. Відсутність обмежень означає, що розробник може реалізувати все саме так, як він бажає, використовуючи величезну кількість зовнішніх бібліотек та додатків, що робить фреймворк гнучким та розширюваним. З іншого боку, Django з його вбудованими функціями та модулями пропонує набагато менше свободи та контролю.

Django – це дуже зрілий фреймворк, перший реліз якого був випущений у 2005 році. Це означає, що він зібрав численні розширення,

плагіни та сторонні додатки, що охоплюють широке коло потреб. Flask є набагато молодшим, був випущеним в 2010 році, тому не має того ж діапазону доступних для нього варіантів.

Flask більше підходить для менших та простіших додатків, тоді як Django призначений для більш великих, більш складних і високо завантажених проектів [7].

Таблиця 2.1

Порівняння фреймворків веб-програмування

Flask	Django
Переваги	
Мінімалістичний, але потужний. Не потрібно багато коду для запуску серверу та початку роботи.	Адміністративна панель. Містить панель адміністрування та автентифікації.
Багато ресурсів та документації. Один з найпопулярніших фреймворків для веб-розроблення мовою Python; багато додаткових модулів та бібліотек	Чітка MVC організація. Забезпечує модульність та повторне використання коду.
Гнучкість. У стандартній комплектації містить мінімальні необхідні функції, залишаючи розробнику рішення про реалізацію окремих компонентів	Легке керування базами даних. Автоматичне створення таблиць та полів, автоматичні міграції.
Відсутність ORM. Розробники мають змогу використовувати «чистий» SQL, або за необхідністю – ORM з модулів.	Зріле ПЗ з великою кількістю плагінів.
Недоліки	
Відсутність асинхронності. Flask явно не призначений для обробки асинхронного програмування.	Масивність. Для малих проектів може бути невиправдано важким.

Продовження табл. 2.1

Налаштовування великих проектів потребує знань фреймворку та навичок програмування.	Складна маршрутизація. Для побудови маршрутів розробнику необхідно знати регулярні вирази.
	Погана документація. Багато детальної документації, яка, однак, не покриває реальні проблеми та не містить прикладів.

Отже, для розроблення було обрано мікро-фреймворк Flask, оскільки основні функції додатку не потребують складної архітектури системи та не передбачають наявності складної бази даних.

2.4. Огляд існуючих наборів даних рукописного тексту

Перед початком розроблення будь якого ML-застосунку необхідно чітко визначити, на яких даних будуть натреновані моделі та скільки коштує ці дані зібрати, оскільки іноді збір даних може займати багато часу, зусиль та грошей, а очікуваний прибуток буде набагато меншим.

Довгий час вважалося, що запорукою успіху моделі є велика кількість тренувальних даних. Насправді ж, більш важливою є якість тренувальних даних, ніж їх кількість. На щастя, існує декілька вже підготовлених, очищених та розмічених датасетів рукописного тексту. Отже, перед початком розроблення алгоритму розпізнавання було проаналізовано наступні датасети.

2.4.1. Cyrillic-oriented MNIST

Ці дані були зібрані методом crowd-sourcing, тобто залученням багатьох різних людей з різних частин світу за домопогою мережі Інтернет з подальшою перевіркою та фільтрацією відповідальною особою – модератором [8].

Переваги:

1. Містить зображення 33 літер української абетки та 26 літер англійської;
2. Більше, ніж 28 000 зображень розміром 278x278 пікселів;
3. Для цього датасету було створено API, завдяки якому можна зробити розпізнавання літери, написаної користувачем на тач-скріні.

Недоліки:

1. Не містить слів або речень. Для створення алгоритму розпізнавання рукописного тексту не вистачає лише поодиноких літер, необхідно мати приклади слів, речень та абзаців.
2. Не використовувався в жодних наукових статтях, тобто не можна знайти вдалі моделі та порівняти їх між собою.

Цей датасет найбільш цікавий тим, що він містить багато прикладів літер кирилицею, що могло б сприяти створенню розпізнавання текстів саме українських текстів. Проте одних лише літер не вистачить для побудови алгоритму, оскільки люди можуть по-різному писати слова та робити переходи між літерами. Тож, треба використовувати дані, які б містили багато прикладів такого поєднання літер у слова, а також слів у речення та абзаци. Нажаль, таких датасетів українською мовою ще не існує, отже доведеться використовувати англомовні.

2.4.2. Iam On-line Handwriting

IAM On-Line Handwriting Database (IAM-OnDB) – база даних рукописного вводу, що містить форми рукописного англійського тексту на білому тлі. Вона містить приклади почерку 657 письменників, які зробили внесок у датасет, загальною кількістю 1 539 сторінок сканованого тексту, 5 685 відокремлених та проанотованих речень, 13 353 відокремлених та проанотованих рядків тексту та 115 320 слів [9]. Його було організовано у декілька різних способів, наприклад, по

реченням, по словам, по формам, по письменникам, які його створювали тощо. Таким чином його можна використовувати для різних цілей, як для сегментації текстів, розпізнавання слів окремо, або ж встановлення, чи належать почерки одній людині.

Переваги:

1. Дуже широковідомий датасет, який постійно доповнюється та оновлюється.
2. Безліч наукових статей, які порівнюють результати роботи алгоритмів на цьому датасеті.
3. Хороша організація, яка дозволяє виконати усі під-завдання на одному датасеті, наприклад, сегментація тексту та розпізнавання.

Недоліки:

1. Усі тексти написані без виправлень, з достатньою відстанню між словами та рядками, що рідко зустрічається у реальному житті.

2.4.3. *Bentham*

Рукописи Бентхама – це великий набір документів, написаних відомим англійським філософом та реформатором Джеремі Бентамом (1748-1832).

Транскрипція цієї колекції в даний час здійснюється волонтерами-аматорами. Наразі за допомогою цієї загальнодоступної веб-платформи було переписано більше 6 000 документів. Містить 433 сторінки, 12 834 унікальних слів та 11 473 унікальних рядків. Більшість документів написані англійською мовою, але деякі його частини написані французькою та грецькою. Дані анотовані по словах та по рядках [10].

Цей датасет також іноді використовують у наукових статтях, проте зазвичай для порівняння з Iam On-line Handwriting, при чому більшість алгоритмів показують кращі результати на останньому.

Отже, було обрано Iam On-line Handwriting, оскільки він найбільш повний та популярний. Проте доведеться виконувати додаткову обробку зображень, щоб адаптувати реальні зображення тексту до цього «ідеального» датасету.

2.5. Інструменти машинного навчання

Розглянемо основні бібліотеки для створення моделей машинного навчання та проаналізуємо їх переваги та недоліки.

2.5.1. Tensorflow

TensorFlow – відкрита програмна бібліотека для машинного навчання цілій низці задач, розроблена компанією Google для задоволення її потреб у системах, здатних будувати та тренувати нейронні мережі для виявлення та розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди.

В TensorFlow усі моделі будуються у вигляді графів операцій, або тензорів, після налаштування яких їм необхідно передати дані для виконання [11].

Переваги :

1) статичний обчислювальний граф;
2) має кращу візуалізацію обчислювального графіка, яка йде «із коробки»;
3) хороша підтримка від Google: бездоганна продуктивність, швидкі оновлення та часті нові випуски з новими функціями;
4) можливість виконувати частину графу дає чудовий інструмент відлагодження;
5) масштабування;
6) можливість паралелізації та використання різних апаратних частин;
7) добре підходить для глибоких нейронних мереж;

8) створений для серверів;

9) простіше оптимізувати статичний обчислювальний граф, оскільки він дозволяє попередньо розподіляти буфери, об'єднувати шари, попередньо компілюючи функції;

10) TensorBoard дає змогу діагностувати моделі;

11) добре підходить для великих багатоплатформниих проектів.

Недоліки:

1) недостатньо швидкий;

2) підтримка лише Nvidia GPU;

3) дуже складний синтаксис;

4) відсутність претренованих моделей;

5) низькорівнева бібліотека.

2.5.2. *PyTorch*

PyTorch – бібліотека машинного навчання для мови Python з відкритим сирцевим кодом

PyTorch надає дві основні високорівневі моделі: Тензорні обчислення з розвиненою підтримкою прискорення на GPU та Глибокі нейронні мережі на базі системи autodiff. Тензори не являють собою нічого особливого, це просто багатовимірні масиви.

PyTorch використовує метод автоматичної диференціації. Проводиться запис обчислень, вироблених в прямому напрямку, потім проводиться відтворення в зворотному порядку для обчислення градієнтів. Цей метод особливо корисний при побудові нейронних мереж, так як дозволяє розраховувати диференціальні поправки параметрів одночасно з прямим проходом [12, 13].

Переваги:

1) динамічний обчислювальний граф надає більшої гнучкості;

2) динамічні структури даних в мережі;

3) модульні мережі можуть бути реалізовані окремо одна від одної;

4) компактній код;

5) імперативне програмування;

6) типовий режим за замовчуванням більше схожий на традиційне програмування, що дає можливість використовувати загальні інструменти налагодження як відладчик pdb, ipdb або PyCharm;

7) має багато претренованих моделей та модулів, які можна використовувати;

8) підтримка розподілених обчислень.

Недоліки:

1) не має інтерфейсів для моніторингу та візуалізації, як, наприклад, Tensorboard;

2) не підходить для роботи на сервері;

3) відносно новий фреймворк, який краще за все підходить для прототипування малих проектів.

Після аналізу існуючих бібліотек та фреймворків, було обрано TensorFlow, оскільки він дуже швидко розвивається та дуже добре підтримується. Крім цього, в ньому є дуже цінний інструмент візуалізації та діагностики Tensorboard та в багатьох наукових статтях використовується саме він.

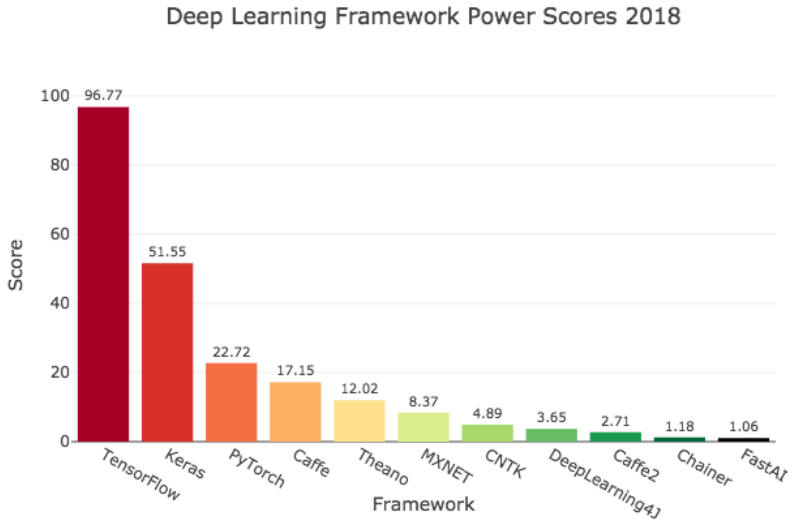


Рис. 2.1. Популярність інструментів машинного навчання

2.6. Аналіз способів розпізнавання рукописного тексту.

Процес розпізнавання рукописного тексту складається із наступних підзавдань:

1) виокремлення сторінки на фотографії, встановлення меж тексту;
2) обробка зображення, збільшення контрастності та товщини ліній;
3) компенсація освітленості, висвітлення темних ділянок;
4) вирівняння нахилу тексту (deslanting);
5) вирівняння нахилу рядку (skew correction);
6) сегментація сторінки на рядки;
7) сегментація рядків на слова;
8) розпізнавання слів;
9) коригування розпізнаних слів та речень за допомогою технік обробки природньої мови.

2.6.1. Сегментація тексту

Існують алгоритми [14], які дозволяють одразу на сторінці знаходити слова та розпізнавати їх, пропускаючи пункти 6-7. Обраний датасет дозволяє натренувати таку модель, але реальні дані будуть дуже відрізнятись від тренувальних структурою. В такому випадку можна застосувати техніку transfer learning, тобто спочатку навчити модель на одних даних, а потім довчити на реальних. Проте оскільки процес збору даних займає багато часу та зусиль, треба обрати більш стійкі алгоритми. Таким чином, для вирішення проблеми сегментації тексту буде використано статистичні підходи.

В процесі аналізу існуючих рішень сегментації тексту, було знайдено модуль, який містить реалізацію статей [15, 16]. Не зважаючи на те, що ці статті були написані у 2007 та 1999 році відповідно, вони і досі є конкурентоспроможними, показують хороші результати та досить швидко працюють.

Модуль також містить реалізацію таких алгоритмів:

1) сканування документів, знаходження контура зображення, використовуючи трансформацію по чотирьох точках;
2) бінаризація Niblack, Sauvola та Wolf;
3) компенсація освітлення за статтею [17];
4) сегментація за алгоритмом [15, 16];
5) вирівняння нахилу тексту за алгоритмом [18].

Крім того, усі вищевказані алгоритми реалізовані мовою C++ та використовують OpenCV, завдяки чому вони працюють набагато швидше.

2.6.2. Алгоритми розпізнавання тексту

Оскільки готових модулів розпізнавання рукописного тексту не існує, було проаналізовано алгоритми машинного навчання. Вже декілька десятків років вчені створюють різні моделі для розпізнавання рукописного тексту. Історично цю проблему формулювали як проблему співставлення послідовностей: послідовність властивостей, сформованих на основі вхідного зображення, співставляється з послідовністю літер; для цього використовувались Приховані Моделі Маркова. Основним недоліком таких моделей є те, що вони не використовують контекст даних у повній мірі через основну марковську властивість, згідно з якою кожне спостереження залежить лише від поточного стану та не залежить від попередніх. Це обмеження було подолано за допомогою таких мереж, як RNN та LSTM, які здатні «запам’ятовувати» досить довгі послідовності. Проте справжній прорив відбувся з винайденням Connectionist Temporal Classification (CTC) [19] та їх поєднанням з рекуррентними мережами [20].

Порівняльна характеристика декількох моделей розпізнавання рукописного тексту наведена у табл. 2.2.

Усі ці моделі були натреновані на IAM датасеті, тож можна порівняти їх якість. Використано дві метрики: Character Error Rate (CER) та Word Error Rate (WER), які визначаються за формулами (1) та (2).

$$CER = \frac{N}{M}, \tag{1}$$

де N – кількість неправильно класифікованих символів у реченні, M – загальна кількість символів у реченні.

$$WER = \frac{L}{P}, \tag{2}$$

де L – кількість неправильно визначених слів у реченні, P – загальна кількість слів у реченні.

Отже, для даного дипломного проекту було обрано модель, яка складається з декількох згорткових шарів (CNN), декількох рекурентних шарів (RNN), а саме шар LSTM, та шаром Connectionist Temporal Classification (CTC).

Таблиця 2.2

Порівняльна характеристика моделей

Model	CER(%)	WER(%)	Avg Runtime(min)	#params(Mi)
HMM [21]	30.6	N/A	N/A	N/A
1D-LSTM [22]	8.2	25.4	3.8	9.3
2D-LSTM [22]	8.3	27.5	24.3	2.6
CNN-1DLSTM-CTC[20]	6.2	20.2	N/A	N/A

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ

3.1. Структура програмних засобів

Розглянемо детально структуру програмних засобів, а саме загальну структуру системи та структуру бази даних.

3.1.1. *Загальна структура*

Програмне забезпечення представляє собою клієнт-серверну архітектуру, зображену на рис. 3.1.

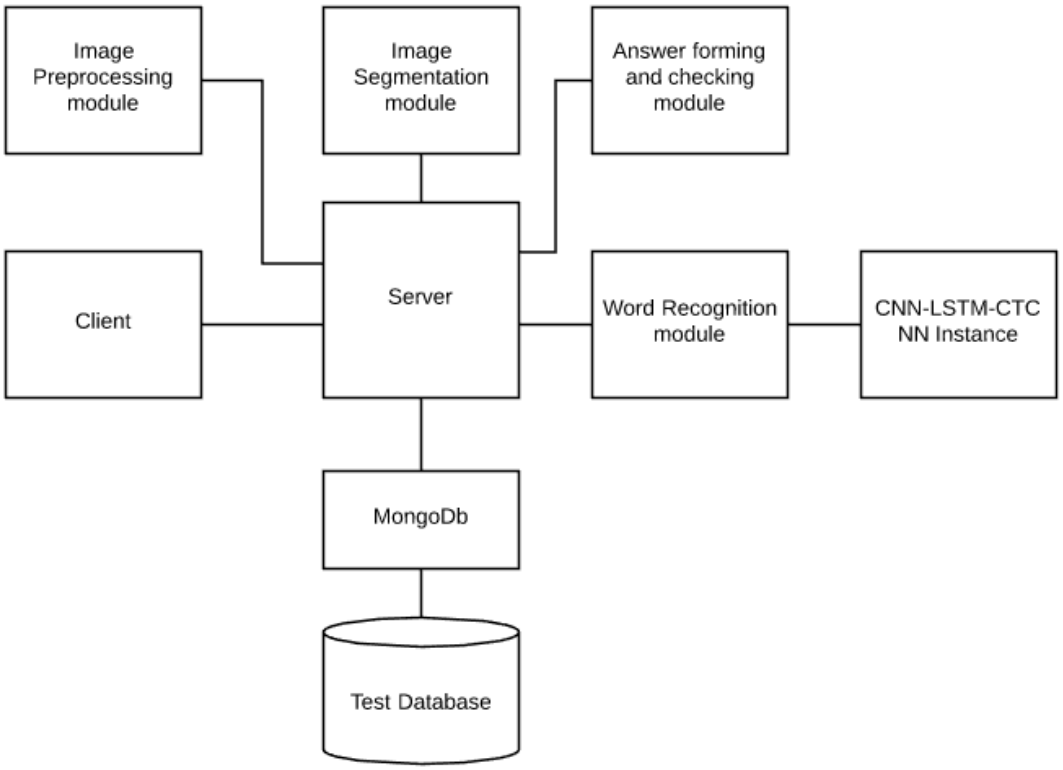


Рис. 3.1. Структурна схема системи

Клієнт надає інтерфейс користувачеві для взаємодії із сервером, надсилає запити на сервер, отримує від нього відповіді та завантажує результати у веб-браузер.

Сервер отримує запити від клієнта, оброблює їх, формує веб-сторінку та повертає її клієнту.

Крім того, він виконує наступні функції:

1) взаємодія з базою даних, яка містить інформацію про тести, створені користувачем;
2) взаємодія з модулем попереднього оброблення зображень;
3) взаємодія з модулем сегментації тексту;
4) взаємодія з модулем розпізнавання тексту;
5) взаємодія з модулем формування відповіді та виставлення оцінки.

3.1.2 *Структура бази даних*

Інформація про зареєстрованих користувачів, про створені ними тести та про результати проходження тестів студентами зберігається у одній базі даних MongoDb. Її загальна структура зображена на рис. 3.2.

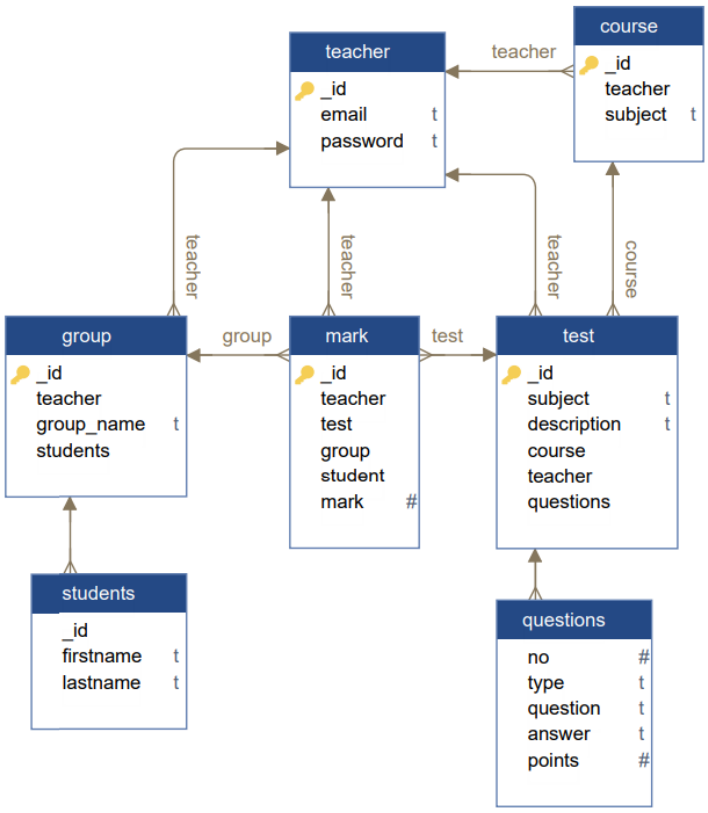


Рис. 3.2. Структура бази даних

Колекція Teacher містить інформацію про користувача (вчителя), його унікальний ідентифікатор, пошту та пароль для авторизації.

Колекція Course має такі атрибути:

- унікальний ідентифікатор курсу;
- унікальний ідентифікатор вчителя, що його створив;
- назву дисципліни.

Якщо при створенні нового тесту вчитель не вказав назву курсу, до якого він належить, буде створено новий курс з назвою «Інше», до якого автоматично потрапляють такі тести.

Колекція Test містить наступні атрибути:

- унікальний ідентифікатор тесту;
- унікальний ідентифікатор курсу, до якого він належить;
- опис тесту;
- дата створення;
- список питань.

Питання не винесені у окрему колекцію, а представляють собою список вкладених об’єктів з наступними полями:

- порядковий номер питання;
- тип питання (відкритий/закритий/послідовність/набір);
- текст питання (не обов’язкове поле);
- правильна відповідь;
- кількість балів, що нараховується за правильну відповідь.

Колекція Group містить такі атрибути:

- посилання на вчителя, який створив цю групу;
- номер групи;
- список студентів.

Сутність «студент» зберігається в базі даних не окремою колекцією, а вкладеними документами, оскільки використовується в розробленому програмному забезпеченні лише разом з групою

безпосередньо. Але для однозначності їм було додано унікальний ідентифікатор.

Атрибути сутності «Студент»:

- унікальний ідентифікатор;
- ім’я;
- прізвище.

Зв’язки між сутностями винесено у табл. 3.1.

Таблиця 3.1

Зв’язок між сутностями у базі даних

№	Сутність	Сутність	Тип зв’язку
1	Teacher	Course	Один до багатьох
2	Teacher	Test	Один до багатьох
3	Test	Questions	Один до багатьох
4	Test	Mark	Один до багатьох
5	Teacher	Group	Один до багатьох
6	Group	Students	Один до багатьох
7	Group	Mark	Один до багатьох

3.2. Алгоритм оброблення вхідного зображення

Оскільки зображення у датасеті, обраному для тренування нейронної мережі, містять чіткий, контрастний рукописний текст з великими інтервалами між рядками та словами, необхідно трансформовувати кожне зображення таким чином, щоб воно будо максимально схоже на зображення із тренувального датасету. Для тестування роботи цієї частини проекту було зібрано дані реальних тестів з англійської мови. Нажаль, вдалося зібрати лише вже перевірені роботи, які містять коригування червоним кольором. На цьому етапі довелося прибирати червоний колір із зображення за допомогою засобів OpenCV [23].

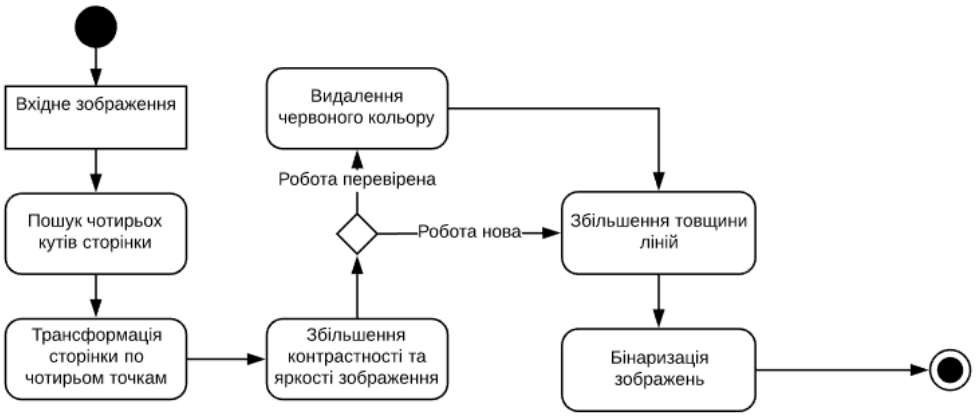


Рис. 3.3. Алгоритм попереднього оброблення зображення

Алгоритм оброблення зображення наведений на рис. 3.3 та містить наступні кроки:

1. Встановлення границі сторінки, фіксація координат чотирьох кутів. Шукається опукла оболонка за допомогою функції OpenCV convexHull, а потім апроксимується до прямокутника функцією approxPolyDP.
2. Трансформування сторінки по чотирьох точках (див. підрозділ 3.3) для отримання фотографії у вигляді знятої під кутом 90° до площини зображення.
3. Збільшення контрастності та яскравості. Це необхідно для того, щоб отримати більш чітку різницю між текстом та папером, а також для того, щоб частково прибрати можливі дефекти фону, наприклад, заломи паперу, розмітку у клітинку, просвічування тексту зі зворотного боку тощо).
4. Видалення червоного кольору. Цей крок виконується лише для перевірених червоним кольором робіт, а також для робіт на папері з червоною смужкою відокремлення полів.
5. Збільшення товщини ліній. Після обробки у попередніх кроках деякі лінії стають переривчастими та тонкими. Їх необхідно потовщити, щоб відновити цілісність слів, а також зробити їх максимально схожими на тренувальні дані.

6. Бінаризація зображення. Використовується метод бінаризації Wolf, що дозволяє представити зображення як двовимірний масив нулей та одиниць, який після цього стає вхідним вектором для алгоритму сегментації.

Результат оброблення зображення наведено на рис. 3.4.

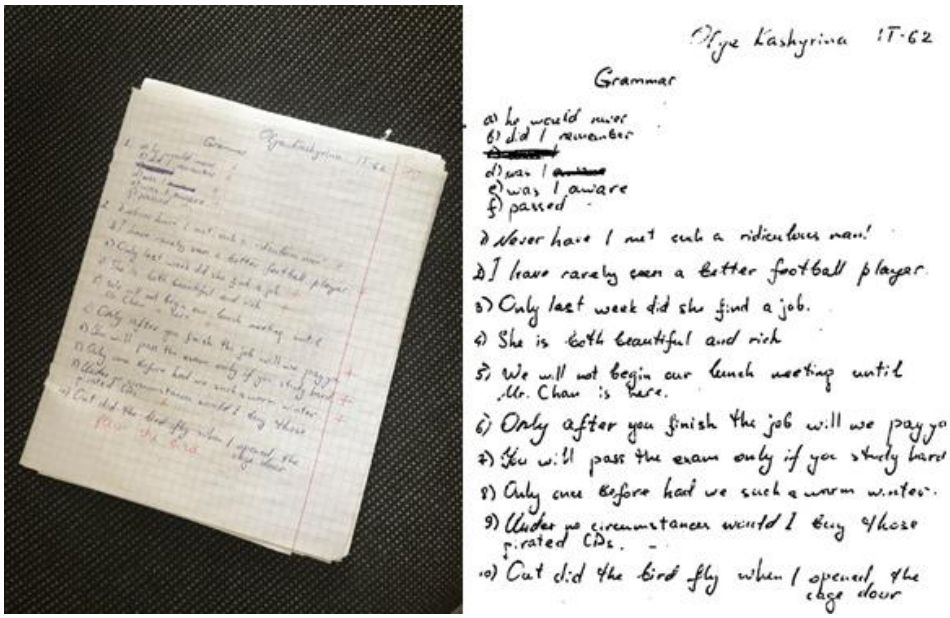


Рис. 3.4. Результат оброблення зображення

3.3. Алгоритм трансформації по чотирьох точках

Якщо користувач зробить фото письмової роботи під кутом до площини зображення, відмінним від 90°, алгоритм розпізнавання слів на зображенні не спрацює, оскільки з точки зору комп’ютера одне й те саме слово, написане з різним нахилом, – це різні слова. Цю проблему можна вирішувати на етапі формування тренувальних даних, додаючи усі можливі викривлення до зображення, щоб алгоритм машинного навчання починав застосовуватись до такого роду відмінностей та правильно класифікував навіть дуже нахилені слова.

Проте набагато швидшим та ефективнішим методом є застосування трансформації з перспективою (Perspective

Transformation) [24]. Для реалізації цієї трансформації було використано функції з бібліотеки OpenCV.

Для застосування трансформації необхідно задати координати чотирьох кутів сторінки, які стають відомими після першого етапу оброблення, та координати кутів нового зображення, які визначаються наступним чином.

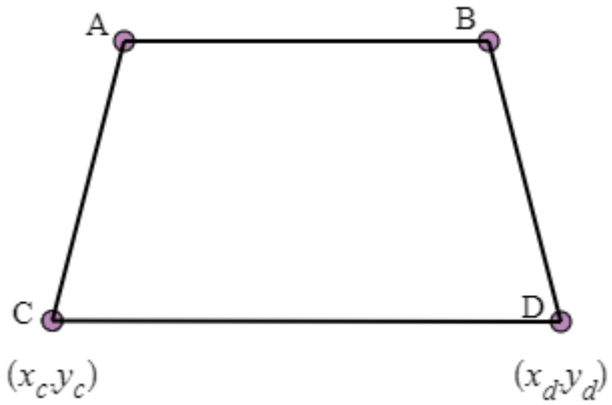


Рис. 3.5. Схематичне позначення сторінки, сфотографованої під кутом.

Нехай A – верхній лівий кут сторінки, B – верхній правий, C – нижній лівий, D – нижній правий, як позначено на рис. 3.5. Їх координати відповідно будуть $(x_a, y_a), (x_b, y_b), (x_c, y_c), (x_d, y_d)$.

Знаходимо довжину сторін за формулами (3) – (6):

$$AB = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}, \tag{3}$$

$$CD = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}, \tag{4}$$

$$CA = \sqrt{(x_a - x_c)^2 + (y_a - y_c)^2}, \tag{5}$$

$$BD = \sqrt{(x_d - x_b)^2 + (y_d - y_b)^2}. \tag{6}$$

Знаходимо ширину та висоту нового зображення наступним чином:

$$W = \max(AB, CD), \tag{7}$$

$$H = \max(CA, BD). \tag{8}$$

Після цього координати кутів нового зображення визначаються наступним чином. Верхній лівий кут має координати $(0; 0)$, верхній правий кут має координати $(W; 0)$. Нижній лівий кут має координати $(0; H)$, нижній правий – $(W; H)$.

Після цього застосовується функція getPerspectiveTransform [25] для отримання матриці переходу та функція warpPerspective для застосування цієї матриці переходу та отримання нового зображення. Схема алгоритму наведена на рис. 3.6.

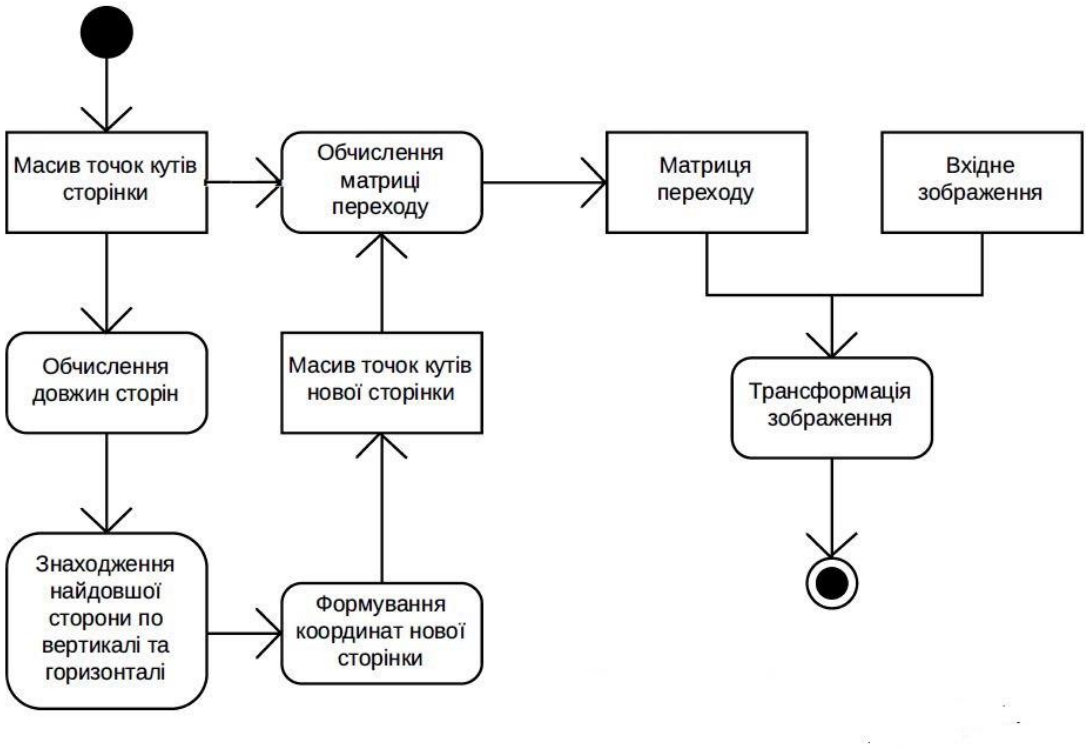


Рис. 3.6. Алгоритм трансформації зображення по чотирьох точках

3.4. Алгоритм розпізнавання слів на зображенні

Модель машинного навчання, за допомогою якої класифікується зображення,– це нейронна мережа, яка містить п’ять згорткових шарів, два рекурентних шара, та після цього застосовується CTC – Connectionist Temporal Classification – шар [26]. Формально цю модель можна представити як функцію, яка ставить у відповідність зображенню, тобто

матриці розміром $W \times H$, послідовність символів $(c_0, c_1, \ldots, c_L)$ довжиною від 0 до L.

Таким чином, розпізнавання тексту відбувається символ за символом, що дозволяє класифікувати слова, яких раніше не було у тренувальних даних, імена та слова, написані з помилками.

Архітектуру нейронної мережі схематично зображено на рис. 3.7.

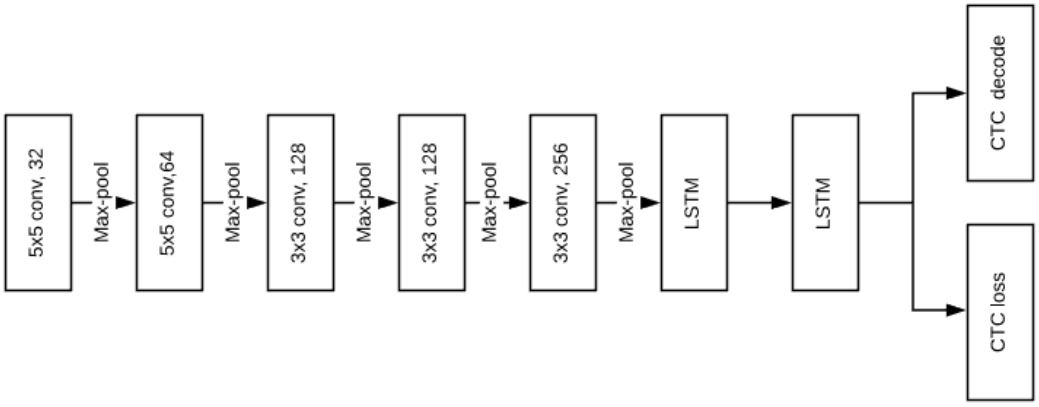


Рис. 3.7. Архітектура нейронної мережі

На вхід алгоритму подається двійкове зображення розміром 128×32. Оскільки не всі зображення слів матимуть такий розмір, необхідно змінювати розмір, поки або висота зображення не стане 32, або ширина не стане 128. Після цього зображення доповнюється білими пікселями до розміру 128×32.

Спочатку зображення поступає на вхід згортковим шарам. Ці п’ять шарів вчаться знаходити релевантні ознаки у зображенні. Кожен шар складається з трьох операцій. Спочатку застосовується фільтр розміром 5×5 для перших двох шарів та 3×3 для останніх трьох. Потім застосовується нелінійна функція активаціїї RELU, яка визначається як

$$f(x) = \max(0, x), \tag{9}$$

де x – вхідне значення нейрона.

Після цього йде агрегувальний шар (pooling layer), який зменшує розмірність вхідного зображення. В якості шару агрегації в цій моделі використано max-pool, який для кожної ділянки розміру 4x4 залишає

лише максимальне значення, та таким чином знаходить лише узагальнену версію властивостей вхідного зображення. Це дуже корисно, оскільки невеликі зміни ознак вхідних даних, виявлених згортковим шаром, не вплинуть на результуючу матрицю ознак.

В результаті після проходження цих п’ятьох шарів ми отримуємо матрицю ознак розміром 32×256, яка передається у рекурентні шари. Це матриця, що містить 32 послідовності із 256 ознак за кожен квант часу.

В якості рекурентних шарів було обрано шари LSTM, оскільки вони здатні поширювати інформацію на довгі відстані та вчити довгі послідовності символів. На виході рекурентних шарів отримуємо матрицю розміром 32×80, де 32 – кількість квантів часу, а 80 – це кількість унікальних символів у IAM dataset (79) + 1 додатковий символ, що необхідний для подальшої Connectionist Temporal Classification.

Під час тренування CTC отримує матрицю розміром 32×80 та правильні відповіді, який саме текст міститься на зображенні, та рахує функцію втрат (CTC loss). Під час передбачення, CTC отримує лише матрицю властивостей та декодує її у текст.

В якості оптимізатора було обрано RMSProp. Результат натренованої моделі на валідаційному наборі даних: CER = 10.63%.

3.5. Алгоритм формування масиву відповідей

Для подальшого порівняння відповідей з еталонною відповіддю, формується масив, у якому містяться відповіді на кожне запитання у рядку. Отже, після оброблення зображення виконується сегментація зображення на рядки. Кожен рядок після цього ділиться на слова. Зазначимо, що на цьому етапі нам відомо порядок рядків у тексті та порядок слів у рядку. Отримавши масив зображень кожного слова для поточного рядка, починаємо розпізнавання. Якщо слово на зображенні – число і воно стоїть першим у рядку, рахуємо, що це початок відповіді на питання під номером цього числа, якщо воно стоїть другим, або далі –

додаємо його до відповіді як звичайний текст. Якщо було визначено початок нової відповіді, то усі слова, що йдуть за цим числом, відносяться до відповіді, поки не буде знайдено нове число на початку рядка. В результаті роботи алгоритму створюється масив, кожний елемент якого – відповідь студента на відповідне запитання, за яким наступним кроком відбуватиметься перевірка тесту. Алгоритм зображено на рис. 3.8.

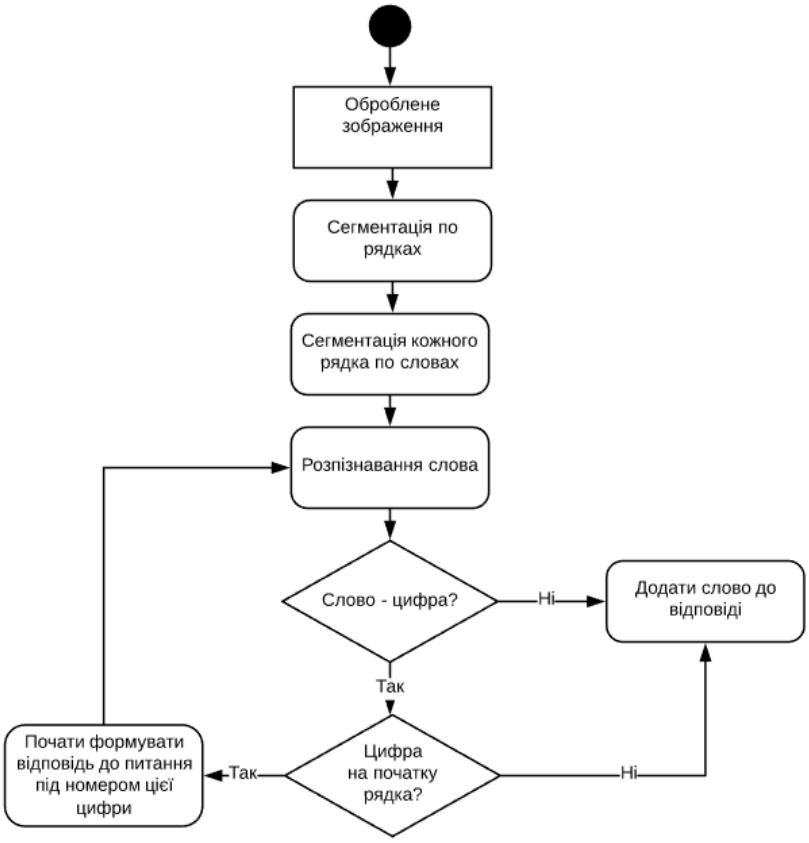


Рис. 3.8. Алгоритм формування масиву відповідей

3.6. Алгоритм перевірки відповідей

Після оброблення зображень, сегментації тексту, розпізнавання слів, та формування масиву відповідей, починається перевірка тесту. У системі передбачено наступні типи питань: відкрите, закрите з однією правильною відповіддю, встановлення послідовності та закрите з декількома правильними відповідями, або набір. Тип запитання «набір»

може бути двох видів: за одну правильну відповідь нараховуються бали, або ні.

Найпростіший випадок – закрите питання з однією відповіддю – перевіряється шляхом порівняння написаних літер у нижньому регістрі.

Якщо закрите питання містить декілька правильних відповідей (тип запитання «набір»), можливі два сценарії. Якщо при створенні питання вчитель вказав, що необхідно визначити усі правильні відповіді, виконуються наступні кроки:

1) літери зводяться до нижнього регістру;
2) сортуються за абеткою;
3) конкатенуються в одне слово;
4) співставляються з правильною відповіддю.

Якщо викладач має намір нараховувати бали за кожну правильно встановлену відповідь, то процес перевірки наступний:

1) літери зводяться до нижнього регістру;
2) рахується перетин між множинами правильних відповідей та фактичних;
3) рахується кількість правильних відповідей.

Якщо питання на встановлення послідовності, то це питання перевіряється аналогічно до питання типу набір, коли необхідно визначити усі правильні відповіді.

Після перевірки закритих питань, студенту одразу нараховується кількість балів, встановлена викладачем за ці питання.

Для перевірки відкритих текстів використовується метрика «косинусова відстань» та Tf-idf векторизаці із пакету sklearn. Алгоритм перевірки наступний:

1) створити об’єкт класу TfidfVectorizer;
2) за допомогою виклику функції fit_transform, отримати два вектори довжиною n, що представляють правильну та фактичну відповідь;

3) сформувати матрицю $A_{2\times n}$, перший рядок якої – вектор правильної відповіді, другий – вектор фактичної;
4) обрахувати нову матрицю як добуток матриці А і транспонованої матриці А за формулою (10).

$$C = A \cdot A^T \tag{10}$$

Матриця С має наступний вигляд:

$$\begin{pmatrix} 1 & s \\ s & 1 \end{pmatrix}, \tag{11}$$

де s – міра схожості правильної відповіді та фактичної. Якщо відповіді повністю співпадають, s = 1. Якщо в них немає нічого спільного s = 0. Відкриті відповіді не оцінюються, проте викладача інформується про рівень схожості.

4. АНАЛІЗ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

4.1. Особливості реалізації серверної частини

Серверна частина додатку реалізована за допомогою мікро-фреймворку Flask, який «з коробки» містить мінімальний функціональні можливості та дає змогу розробнику розширювати його так, як необхідно. В цьому випадку було реалізовано модель MVC (Model-View-Controller). Це архітектурний шаблон проектування користувацьких інтерфейсів, основною метою якого є відокремлення внутрішньої структури даних від їх зовнішнього представлення та підвищення модульності коду.

Модуль моделей (models.py) – центральний компонент MVC [27]. Він відповідає за керування даними веб-додатку за допомогою ORM бібліотеки MongoEngine, яка дозволяє маніпулювати даними з бази даних MongoDb.

В цьому модулі міститься опис усіх класів, які відповідають колекціям та документам бази MongoDb, а саме:

- User – колекція користувачів (викладачів);
- Course – колекція усіх курсів усіх викладачів;
- Question – вбудований документ, який міститься у колекції Test;
- Test – колекція тестів;
- Mark – колекція студентських оцінок.
- Student – вбудовані документи, що містяться у колекції курсів.

Контролер – це певна абстракція, що складається з трьох частин:

- Ініціалізація: створення екземпляру Flask додатку, налаштування необхідних конфігурацій, підключення бази даних.
- Маршрутизація : визначення URL-шляхів, за якими сервер завантажує певні сторінки за допомогою декораторів у мові Python. Кожен маршрут асоціюється з контролером, точніше, з певною дією контролера. Під час виконання цієї дії

використовуються моделі для отримання необхідних даних з бази даних, які після цього надсилаються до модуля представлень для завантаження.

- Виконання: запуск веб-додатку.

В файлі routes.py містяться наступні методи маршрутизації, які пов’язані з Model та View:

- login() – авторизація користувача у системі;
- register() – реєстрація користувача, якщо такого ще не існує. Пароль зберігається у вигляді хешу sha256;
- logout() – вихід користувача з системи;
- courses() – отримання усіх існуючих курсів поточного користувача;
- tests(coursename) – отримання усіх існуючих тестів в рамках курсу coursename поточного кристувача;
- groups() – отримання інформації про усі групи поточного користувача;
- new_test() – створення та збереження нового тесту;
- marks(group) – отримання оцінок кожної групи;
- check(testid) – перевірка тесту;
- reports() – перегляд графіків успішності;
- create_plot(plot_type, group, test) – створення графіку типу plot_type для певної групи з певного тесту;
- get_marks_data(group,test) – формування даних для побудови графіків;
- preprocess(image) – оброблення та сегментація зображення;
- infer(words) – розпізнавання слів на зображеннях;
- form_answer(words) – формування масиву відповіді;
- check(testid, answer) – перевірка правильних відповідей;

Модуль представлення – це абстракція, що відповідає за користувацький інтерфейс, який отримує дані з моделі від контролера та

визначає, як їх показувати. Фреймворк Flask для цього надає двигун шаблонів Jinja2 для генерації HTML-сторінок, який також надає можливість наслідування шаблонів. Так, базовим шаблоном є верхня навігаційна панель, описана у base.html. Її наслідує ліва навігаційна панель з файлу dashboard.html, яку наслідують усі інші. Дизайн та вміст усіх сторінок описаний у підрозділі 4.2.

4.2. Дизайн та вміст сторінок

Для створення користувацького інтерфейсу було використано HTML-бібліотеку Bootstrap, оскільки вона надає цілий ряд засобів для швидкого створення сучасних інтерфейсів.

У верхній частині сторінки розташована навігаційна панель авторизації, описана у файлі base.html, яка наслідує базову сторінку Bootstrap, що необхідно для роботи цієї бібліотеки. Для неавторизованого користувача панель містить лише опції входу та реєстрації (рис. 4.1).

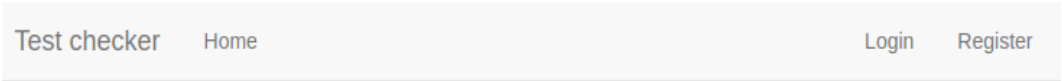


Рис. 4.1. Навігаційна панель для неавторизованого користувача

Якщо користувач зареєструвався та увійшов у систему, на цій панелі міститься кнопка виходу з системи та кнопка Dashboard (рис. 4.2), перейшовши за якою користувач отримує доступ до усіх функцій системи та побачить основне меню.

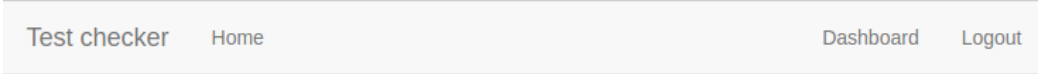


Рис. 4.2. Навігаційна панель для авторизованого користувача

При натисканні на кнопку Login, відкривається форма входу у систему, яка містить лише два основних поля: пошта та пароль, а також прапорець «Запам’ятати мене», поставивши галочку на який, користувач залишиться авторизованим у системі (рис. 4.3). Для незареєстрованих користувачів є посилання на форму реєстрації (рис. 4.4). Ця форма доступна також при натисканні на кнопку Register у верхній навігаційній панелі та відрізняється від форми входу лише додатковим полем підтвердження паролю. В обох формах пароль при введенні маскується за допомогою зірочок. Якщо на формі реєстрації паролі не співпадають, або якщо у формі входу введено неправильний пароль, користувач побачить повідомлення про це. При спробі зареєструватись під поштою, яка вже існує в системі, також висвітиться помилка з проханням обрати іншу пошту.

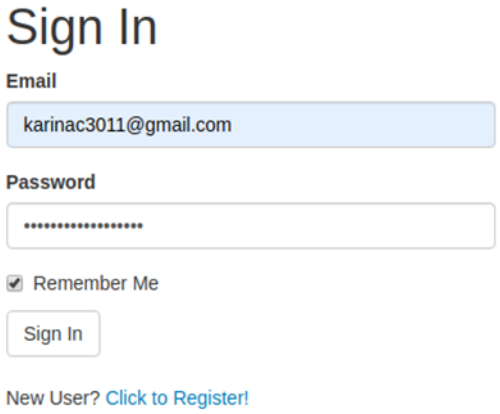


Рис. 4.3. Форма входу

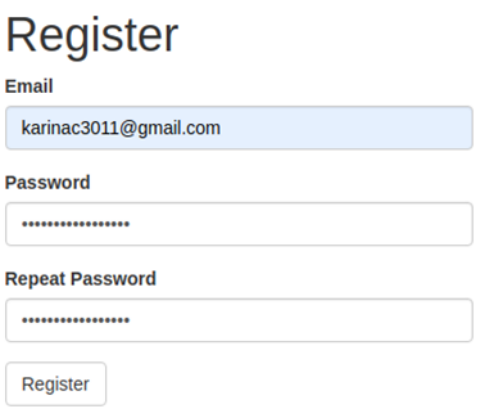


Рис. 4.4. Форма реєстрації

Після входу у систему, користувач бачить основну панель навігації (рис. 4.5), яка описана у файлі dashboard.html і яка наслідує сторінку з панеллю авторизації base.html. Усі інші сторінки системи наслідують сторінку dashboard.html, тим самим створюючи чотири рівні наслідування шаблонів. Таким чином зберігається єдиний стиль сторінок та зменшується кількість коду.

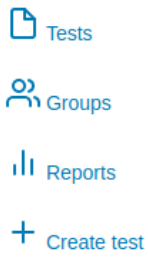


Рис. 4.5. Основна навігаційна панель

Панель навігації містить основні елементи управління та дозволяє мінімізувати кількість проміжних дій при роботі із системою. Серед пунктів меню є посилання на сторінку тестів, сторінку списків груп, сторінку статистики та сторінку створення нового тесту, а також графічні елементи у вигляді іконок feather icons для кожної вкладки для швидкої орієнтації користувачів.

Сторінка списку груп містить форму завантаження нового списку з CSV файлу, а також список усіх вже створених груп (рис. 4.6). Форма імпорту групи складається з текстового поля, в яке необхідно ввести номер групи та поля завантаження файлу. Якщо поточним користувачем не було створено жодної групи, замість списку з групами він побачить повідомлення про це.

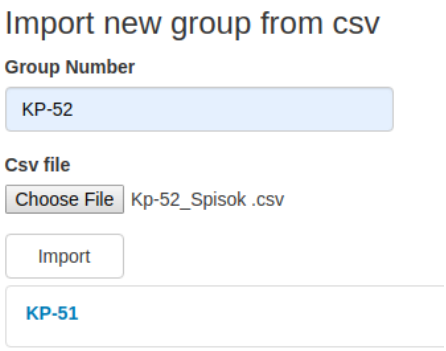


Рис. 4.6. Форма імпорту списку групи

При спробі завантажити файл будь-якого іншого формату, ніж csv, користувач побачить повідомлення про помилку читання файлу. Після

додавання групи, вона автоматично додається до списку як елемент з посиланням, перейшовши за яким, завантажується сторінка поточних оцінок певної групи з усіх тестів.

Сторінка створення нового тесту містить форму з полями назви курсу, опису тесту та групи створення нового питання (рис. 4.7). Ця група містить наступні поля:

1) тип питання, вибір з випадаючого списку з такими варіантами: single-choice, sequence, set, text, які відповідають можливим значенням у базі даних, описаних у підрозділі 3.1;
2) текст питання (необов’язкове текстове поле);
3) правильна відповідь на питання (обов’язкове текстове поле)
4) бали за питання (обов’язкове числове поле, за замовчуванням дорівнює 1);
5) кнопка додавання питання до тесту.

Список питань має форму таблиці з колонками, що відповідають властивостям питання та рядками, що відповідають питанням.

При натисканні кнопки Add question, нове питання одразу показується у списку питань, що розташовується під формою, за допомогою засобів JavaScript, при цьому кнопка збереження тесту переміщується вниз на ширину одного рядка таблиці. При натисканні на кнопку збереження тесту, або ж при спробі піти зі сторінки, з’являється діалогове вікно з підтвердженням дії. Для продовження роботи з системою, користувач має обрати необхідну опцію.

На сторінці тестів показується список усіх курсів, створених поточним користувачем. Якщо при створенні тесту користувач не ввів назву курсу, до якого він належить, такі тести групуються у категорію «Other tests».

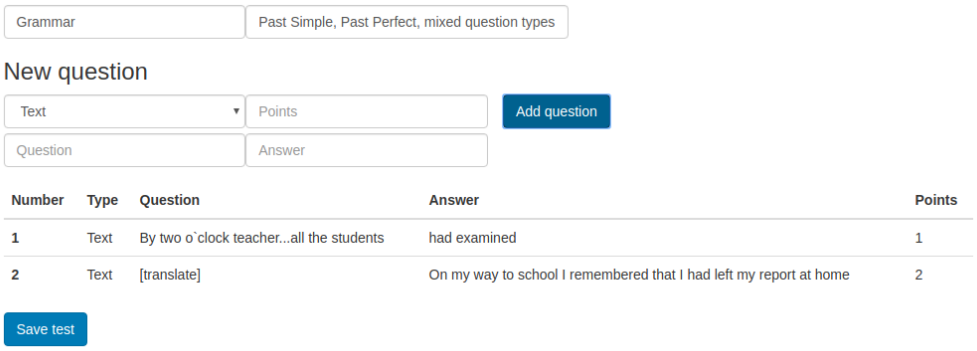


Рис. 4.7. Форма створення нового тесту

Рис. 4.8. Сторінка тестів, згрупованих по курсам

Кожен елемент списку курсів (рис. 4.8) є активним посиланням на сторінку перегляду тестів обраного курсу, яка також містить список створених тестів. В цьому списку міститься номер та опис кожного тесту, а також дата його створення (рис. 4.9).

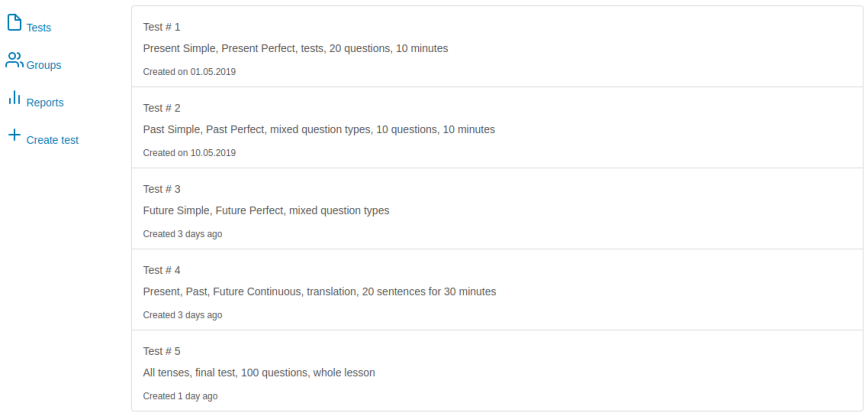


Рис. 4.9. Сторінка тестів з одного курсу

При наведенні курсора на елемент списку, він змінює колір, при натисканні – відкривається сторінка перегляду тесту (рис. 4.10).

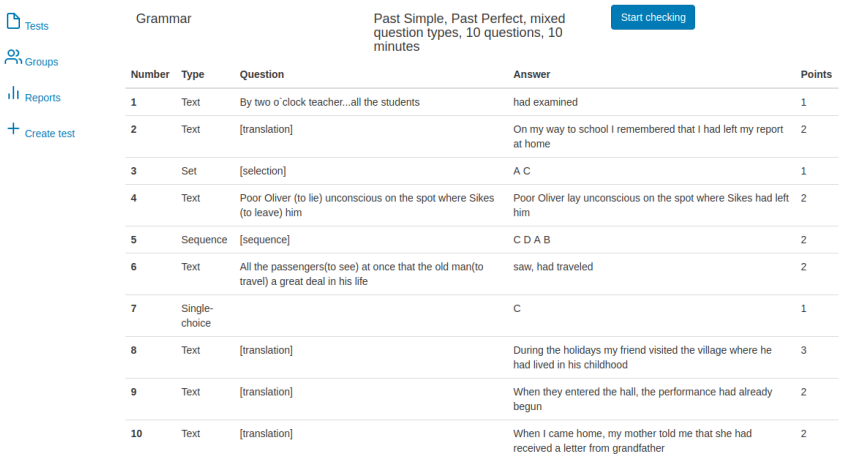


Рис. 4.10. Сторінка перегляду тесту

Вона містить назву курсу, опис тесту, таблицю з питаннями , а також кнопку «Start checking», яка веде на сторінку перевірки обраного тесту.

Сторінка перевірки тесту (рис. 4.11) містить наступні поля:

1) група – випадаючий список, який містить усі групи, які імпортував поточний користувач;
2) студент – випадаючий список з прізвищами студентів у обраній групі. Значення цього поля змінюються динамічно в залежності від значення, обраного у першому випадаючому списку;
3) файл – кнопка завантаження зображення роботи студента з файлової системи;
4) рекомендована оцінка, заповнюється автоматично системою після перевірки завантаженого файлу;
5) оцінка – фактична оцінка, яку викладач може виставити на основі рекомендованої оцінки;
6) кнопка збереження оцінки.

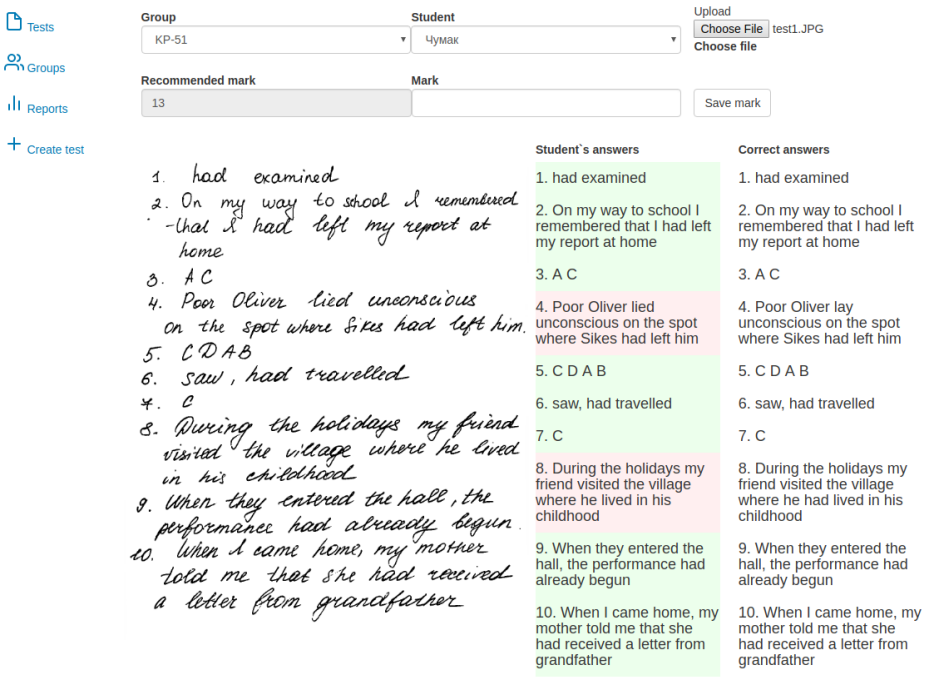


Рис. 4.11. Сторінка перевірки тесту

Після завантаження зображення, під цією формою з'являється зображення роботи після оброблення у чорно-білому кольорі, колонка з відповідями студента, отриманими після розпізнавання зображення, а також колонка з правильними відповідями. Кожна правильна відповідь студента підсвічується зеленим кольором, неправильна – червоним.

При спробі ввести у поле оцінки щось, відмінне від простого цілого числа, користувач побачить повідомлення про помилку.

На сторінці Reports (рис. 4.12) міститься три випадаючих списки:

1) тип графіка: гістограма, або boxplot;
2) група: усі групи поточного викладача, а також опція «All» для перегляду статистики по всіх групах одразу;
3) тест: усі тести поточного викладача, а також опція «All» для перегляду статистики по усім тестам.

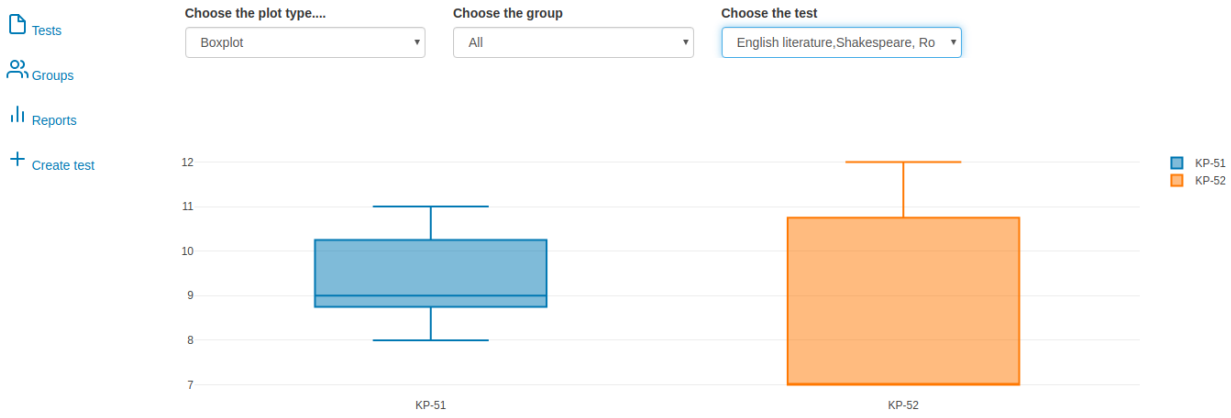


Рис. 4.12. Сторінка побудови графіків

При зміні значення будь-якого з цих полей, графік змінюється динамічно за домопогою інструментів JavaScript та AJAX. Якщо обрано усі групи з певного тесту та тип графіку boxplot, на ньому буде зображено декілька «коробок», відповідно до кожної групи, що дає змогу порівняти групи між собою.

Графіки побудовані за допомогою бібліотеки Plotly, що дозволяє користувачам інтерактивно взаємодіяти з ними, а саме наближувати певну ділянку графіка, завантажувати графік як зображення, бачити інформацію про певні точки при наведенні курсора на них тощо.

4.3. Тестування системи

У зв’язку з тим, що система містить декілька складних алгоритмів, які тісно між собою пов’язані та складають підсистему розпізнавання рукописного тексту, її було протестовано окремо перед початком розроблення web-додатку. Після цього для перевірки якості програмного забезпечення розроблено та проведено набір тест-кейсів димового тестування, описаних у пункті 4.3.2.

4.3.1. *Тестування модуля розпізнавання рукописного тексту*

Розроблений модуль розпізнавання тексту природньо має різну поведінку на різних вхідних зображеннях. Очікувані результати в залежності від різних зображень наведено далі.

Тест-кейс 1

Опис зображення:

1. Робота чітко видима, добре освітлена, не має заломів, плям та інших дефектів.
2. Сторінка знаходиться по центру зображення, до кожного краю однаковий відступ.
3. Кут до площини зображення прямий, або близький до прямого.
4. Робота містить лише довгі речення, що займають усю ширину сторінки.
5. Робота не містить ієрархічних завдань.

Очікувані результати:

1. Знайдено границі роботи. Виділено чотири кути зображення.
2. Відступи до країв зображення прибрано.
3. Усі слова на роботі добре видно, лінії потовщено.
4. Коректна сегментація рядків.
5. Коректна сегментація слів.
6. >70% слів розпізнано правильно.
7. Послідовність слів у вихідному масиві коректна.

Тест-кейс 2

Опис зображення:

1. Робота чітко видима.
2. Сторінка знаходиться по центру зображення, до кожного краю однаковий відступ.
3. Кут до площини зображення відмінний від прямого.

Очікувані результати:

1. Знайдено границі роботи. Виділено чотири кути зображення.

2. Зображення трансформовано так, що кут до зображення прямий.

3. Усі слова на роботі добре видно, лінії потовщено

Тест-кейс 3

Опис зображення:

1. Робота чітко видима.
2. Сторінка знаходиться по центру зображення, відступів до краю немає.
3. Кут до площини зображення прямий.

Очікувані результати:

1. Усі слова на роботі добре видно, лінії потовщено.
2. Коректна сегментація рядків та слів
4. >70% слів розпізнано правильно.
5. Послідовність слів у вихідному масиві коректна.

Тест-кейс 4

Опис зображення:

1. Робота засвічена, чітко проглядається ділянка з більшою освітленістю.
2. Сторінка знаходиться по центру зображення, відступів до краю немає.
3. Кут до площини зображення прямий.

Очікувані результати:

1. Світла пляма скоригована, різниці в освітленні немає
3. Усі слова на роботі добре видно, лінії потовщено.
4. Коректна сегментація рядків та слів
5. Більшість слів розпізнано правильно.
6. Послідовність слів у вихідному масиві коректна.

Тест-кейс 5

Опис зображення:

1. Робота чітко видима.

2. Робота містить дефект фону: просвічується текст зі зворотного боку.
3. Робота містить ієрархічні завдання.
4. Робота містить малюнок.
5. Робота містить закреслення.

Очікувані результати:

1. Дефект фону прибрано
2. Робота контрастна, усі слова на роботі добре видно, лінії потовщено.
3. Коректна сегментація рядків усюди, крім малюнку та порожніх рядках з позначенням номеру завдання.
4. Некоректна сегментація слів на малюнку, закресленнях, порожніх рядках з позначенням номеру завдання.
6. Більшість слів розпізнано правильно.
7. Послідовність слів у вихідному масиві некоректна у зв’язку з ієрархічністю завдань.

Тест-кейс 6

Опис зображення:

1. Зображення поганої якості, погане освітлення, робота знаходиться по центру зображення

Очікувані результати:

1. Сторінка не знайдена, слова не розпізнані

При тестуванні розробленого модуля на тренувальному наборі даних точність розпізнавання склала 80%, на реальних даних точність не вимірювалась, проте очікувано становить менше.

4.3.2. Тестування web-додатку

Для тестування системи в цілому також було виділено декілька тест-кейсів для димового тестування та подано нижче.

Сценарій димового тестування:

1. Зареєструватись у системі.
2. Авторизуватись у системі.
3. Додати нову групу студентів, імпортувавши список з csv файлу.
4. Створити новий тест.
5. Переглянути існуючі тести.
6. Переглянути обраний тест.
7. Розпочати перевірку роботи студента.
8. Виставити студенту оцінку за роботу.
9. Переглянути оцінки обраної групи.
10. Побудувати графіки.

Тест-кейс 1

Опис:

1. Відкрити сторінку реєстрації.
2. Ввести пошту та пароль.
3. Підтвердити пароль.
4. Натиснути кнопку «Зареєструватися».

Очікувані результати:

1. Система показує сторінку реєстрації.
2. Паролі замасковані за допомогою зірочок.
3. Користувач входить до основної сторінки.

Тест-кейс 2

Опис:

1. Відкрити сторінку авторизації.
2. Ввести пошту та пароль.
3. Натиснути кнопку «Ввійти».

Очікувані результати:

1. Система показує сторінку входу.
2. Пароль замаскований за допомогою зірочок.
3. Користувач входить до основної сторінки.

Тест-кейс 3

Опис:

1. Відкрити сторінку перегляду груп.
2. Ввести номер групи у поле.
3. Обрати файл зі списком групи з файлової системи.
4. Натиснути кнопку «Завантажити».

Очікувані результати:

1. Система показує сторінку перегляду груп.
2. Назва обраного файлу показується у відповідному полі.
3. Під формою завантаження з’являється посилання на додану групу.

Тест-кейс 4

Опис:

1. Відкрити сторінку створення тесту.
2. Ввести назву дисципліни.
3. Ввести короткий опис тесту.
4. Обрати тип питання з випадаючого списку.
5. Ввести кількість балів за питання.
6. Ввести правильну відповідь.
7. Натиснути кнопку «Додати питання».
8. Виконувати пункти 4-7 для кожного питання тесту.
9. Натиснути кнопку «Зберегти тест».
10. Натиснути кнопку «Так» у модальному вікні підтвердження.

Очікувані результати:

1. Система показує сторінку створення тесту.
2. Після додавання питання, воно одразу показується у списку питань.
3. Після зберігання тесту з’являється діалогове вікно підтвердження дії.
4. На сторінці тестів є новий тест.

Тест-кейс 5

Опис:

1. Відкрити сторінку тестів.
2. Обрати необхідний курс зі списку курсів.
3. Переглянути тести.

Очікувані результати:

1. Система показує сторінку тестів.
2. Усі тести згруповані по курсах.
3. Тести, які не належать жодному курсу, зберігаються у курсі «Інше».

Тест-кейс 6

Опис:

1. Відкрити сторінку тестів.
2. Обрати необхідний тест.
3. Переглянути вміст тесту.

Очікувані результати:

1. Система показує сторінку тесту.
2. Усі питання тесту показується коректно.

Тест-кейс 7

Опис:

1. Відкрити сторінку перегляду тесту.
2. Натиснути кнопку «Розпочати перевірку».
3. Обрати групу.
4. Обрати студента.
5. Обрати файл з файлової системи.
6. Завантажити роботу.

Очікувані результати:

1. На сторінці перевірки коректно показується форма перевірки тесту.

2. При зміні групи динамічно змінюються варіанти випадаючого списку для поле «Студент».
3. Після завантаження файлу, оброблена версія зображення показується під формою перевірки.
4. Поруч з роботою показуються розпізнані студентські відповіді та коректні відповіді.
5. Неправильні відповіді підсвічені червоним кольором, правильні – зеленим.
7. В поле рекомендованої оцінки з’являється числове значення.

Тест-кейс 8

Опис:

1. Завантажити роботу на сторінці перевірки тесту.
2. Співставити червоні відповіді з правильними.
3. Прийняти рішення про результуючу оцінку.
4. Виставити оцінку у відповідне поле.
5. Натиснути кнопку «Зберегти оцінку»

Очікувані результати:

1. Оцінка збережена у базі даних.
2. Поля очищуються.
3. Результат попередньої роботи програми зникає.

Тест-кейс 9

Опис:

1. Перейти на сторінку списків груп.
2. Обрати групу зі списку, перейти за посиланням.
4. Переглянути оцінки групи.

Очікувані результати:

1. Список груп показується коректно.
2. Оцінки за перевірені тести збережені у таблиці.

Тест-кейс 10

Опис:

1. Перейти на сторінку статистики.
2. Обрати тип графіку.
3. Обрати групу.
4. Обрати тест.
5. Переглянути графіки.

Очікувані результати:

1. Графіки динамічно змінюють при зміні параметрів.
2. При обранні параметру групи як «усі» та параметру графіку «boxplot», будуються графіки для усіх груп поруч один з одним.
3. При обранні групи та тесту, який ця группа не писала, графік відсутній.

4.4. Рекомендації щодо використання розробки

Під час використання веб-сервісу викладачам рекомендується дотримуватись деяких правил. По-перше, завантажуючи роботу студента, краще робити фотографію у добре освітлюваному місці, на хорошу камеру, оскільки інакше система розпізнавання слів може видавати непередбачені результати. Бажано, щоб сторінка, на якій написана робота, була нейтрального кольору, без додаткової розмітки, малюнків тощо. Звичайний папір у клітинку або лінію дозволяється.

По-друге, розроблюючи структуру тесту рекомендується уникати вкладених завдань. Тобто тест може містити безліч запитань, проте не варто групувати їх у окремі завдання, пункти, підпункти тощо.

По-третє, рекомендується наголосити студентам, що слова треба писати на достатній відстані одне від одного, а закриті тесті з декількома відповідями писати в один рядок, без додаткових цифр, також дотримуючись достатньої відстані між ними.

Крім цього студент має писати відповіді послідовно, в одну колонку, не перегортати лист відповіді перпендикулярно, не малювати стрілок послідовності тощо. Виправлення, закреслення, малюнки та будь-які елементи, що відрізняються від слів та цифр, можуть суттєво погіршити якість розпізнавання зображення.

4.5. Рекомендації щодо подальшого вдосконалення

В ході розроблення даного програмного забезпечення було натреновано модель машинного навчання, яка досягає непоганих результатах на IAM датасеті. Тим не менш, в реальності зображення зовсім не схожі на зображення із тренувальних даних, в результаті чого іноді алгоритм погано сегментує сторінку та некоректно класифікує слова. Для подальшого вдосконалення необхідно зібрати власний набір тренувальних даних та зробити його анотацію. Цього не було зроблено у ході розроблення, оскільки це займає дуже багато часу. Одним із способів збору тренувальних даних є краудсорсинг, який і було б доцільно використати у майбутньому. Зауважимо, що зовсім не обов'язково відмовлятись від IAM датасету та вже існуючої моделі. Натомість можна застосувати техніку transfer learning, основною ідеєю якої є тренування моделі на одних даних, а потім використання здатності моделей на перших етапах тренування вчитися розпізнавати низькорівневі абстракції в даних для тренування на іншому наборі даних [28]. Цей підхід дуже часто застосовується у computer vision та має хороші шанси спрацювати і у завданні розпізнавання рукописного тексту.

Крім цього, для покращення розпізнавання можна додати ще декілька шарів у нейронну мережу та збільшити розмір вхідного вектору так, щоб можна було розпізнавати цілі слова, речення та абзаци. Такий підхід потребує більшої кількості тренувальних даних.

В процесі розроблення даного ПЗ модель машинного навчання створювалась не дуже глибокою, оскільки процес тренування таки моделей дуже довгий. Для досягнення хороших результатів рекомендується одночасно поглибити нейронну мережу та виконувати тренування на декількох GPU, а не CPU. Слід врахувати, що хмарні сервіси, такі як AWS [29] та Google Cloud [30] надають можливість використовувати їх відео картки, проте, не безкоштовно, отже тренування може досить дорого коштувати.

Для покращення сегментації зображення рекомендується застосовувати більш сучасні методи навчання з вчителем, тобто використовувати нейронні мережі з тренувальними даними, наприклад, як описано у статті [14].

Для покращення перевірки відкритих запитань варто створити додатковий алгоритм з застосуванням технологій NLP, який може краще співставляти дві відповіді, а також визначати, чи містить текст відповідь на запитання.

ВИСНОВКИ

Метою даного дипломного проекту було створення web-додатку для автоматизації перевірки письмових тестів. Перед початком розроблення було проведено аналіз предметної області, розглянуто проблеми потенційних користувачів, проаналізовано існуючі аналоги, зібрано вимоги до розроблюваного програмного забезпечення та досліджено можливі способи вирішення поставленої задачі. В результаті аналізу існуючих програмних рішень було встановлено, що жодна із запропонованих систем не відповідає вимогам та потребам користувачів, що означає, що розроблення такої системи є актуальним та необхідним. Наведено змістовне обґрунтування вибору технологій розроблення, мов програмування, бібліотек машинного навчання та фреймворків веб-розроблення. Крім цього, пояснювальна записка містить опис розроблених алгоритмів та моделей машинного навчання, опис архітектури системи, приклади її роботи та рекомендації для використання та подальшого розроблення.

Розроблена система здатна розпізнавати рукописний текст з фотографії, формувати масив відповіді та порівнювати його з відповіддю, внесеною викладачем. Для зручності використання було створено мінімалістичний інтерфейс користувача, який забезпечує йому доступ до усіх основних функцій системи.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. History UA: Тести, як засіб перевірки знань школярів [Електронний ресурс] – Режим доступу: https://www.historyua.com/2018/07/27/testy-yak-zasib-perevirky-znan-shkolyariv/
2. Testolog: Нетрадиційні тести [Електронний ресурс] – Режим доступу: http://testolog.narod.ru/Theory4.html
3. eMarketer: Top 5 Stats to Know About US Mobile Usage [Електронний ресурс] – Режим доступу: https://www.emarketer.com/corporate/coverage/be-prepared-mobile.
4. InfoWorld: A developer's guide to the pros and cons of Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.infoworld.com/article/2887974/a-developer-s-guide-to-the-pro-s-and-con-s-of-python.html.
5. Quora: What are pros and cons of Python? [Електронний ресурс] – Режим доступу до ресурсу: https://www.quora.com/What-are-the-pros-and-cons-of-Python.
6. ProsAndCons: Pros and Cons of C Programming Language [Електронний ресурс] – Режим доступу до ресурсу: https://www.prosancons.com/computer/pros-and-cons-of-c-programming-language/R. Manmatha, N. Srimal. Scale Space Technique for Word Segmentation in Handwritten Documents, 1999.
7. PacktHub: Python web development: Django vs Flask [Електронний ресурс] – Режим доступу до ресурсу: https://hub.packtpub.com/python-web-development-django-vs-flask- 2018.

8. Kaggle: Cyrillic MNIST dataset [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/datasets?tagids=14202.
9. Fki. CV and AI: IAM On-Line Handwriting Database [Електронний ресурс] – Режим доступу до ресурсу: http://www.fki.inf.unibe.ch/databases/iam-on-line-handwriting-database.
10. Transcriptorium: Bentham collection [Електронний ресурс] – Режим доступу до ресурсу: http://transcriptorium.eu/datasets/bentham-collection/.
11. DataCamp: TensorFlow Tutorial For Beginners [Електронний ресурс] – Режим доступу до ресурсу: https://www.datacamp.com/community/tutorials/tensorflow-tutorial.
12. PyTorch: PyTorch Tutorials [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/tutorials/.
13. Towards Data Science: PyTorch vs TensorFlow—spotting the difference [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b.
14. G.Axler. Toward A Dataset-Agnostic Word Segmentation Method/ G.Axler, L.Wolf, 2018.
15. M. Arivazhagan .A Statistical approach to line segmentation in handwritten documents / M. Arivazhagan, H. Srinivasan, S. Srihari, 2007.To 3.
16. R. Manmatha, N. Srimal. Scale Space Technique for Word Segmentation in Handwritten Documents, 1999.To 6
17. K.-N. Chen. Efficient illumination compensation techniques for text images/ K.-N. Chen, C.-H. Chen, C.-C. Chang, 2005.To 7
18. A.Vinciarelli. A New Normalization Technique For Cursive Handwritten Words/ A.Vinciarelli, J.Luettin, 2001.To 9

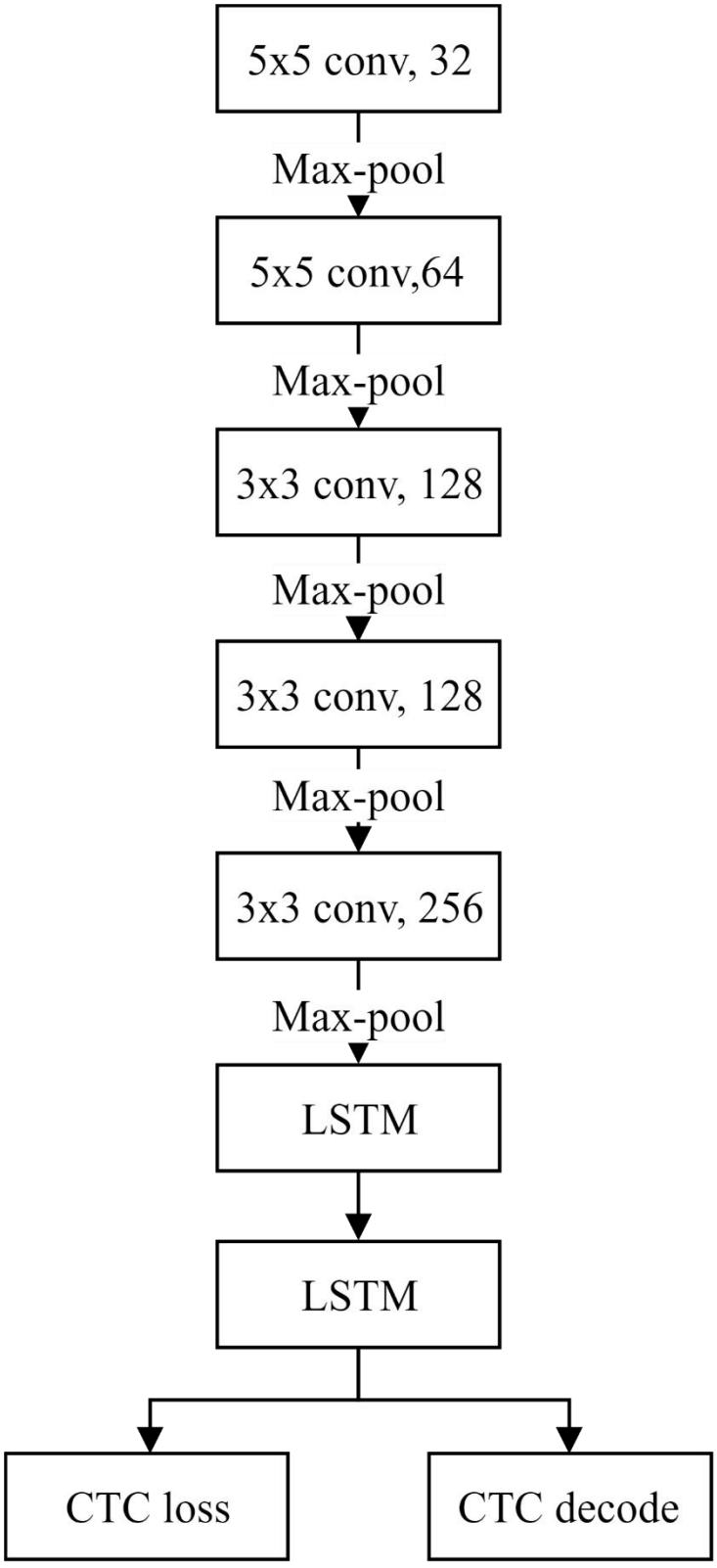
19. A.Graves. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks/ A.Graves, S.Fernandez, F.Gomex, J.Schmidhuber,2006.mongo
20. A.Chowdhury. An Efficient End-to-End Neural Model for Handwritten Text Recognition/ A.Chowdhury, L.Vig, 2018.
21. N.Y.Hammerla. Towards Feature Learning for HMM-based Offline Handwriting Recognition / N.Y.Hammerla, T.Ploetz, G.A. Fink, S.Vajda, 2010.
22. J.Puigcerver. Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition?, 2017.
23. OpenCV: OpenCV Tutorials [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/master/d9/df8/tutorial_root.html
24. PyImageSearch: 4 Point OpenCV getPerspective Transform Example [Електронний ресурс] – Режим доступу до ресурсу: https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/
25. OpenCV: Geometric Transformations of Images [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/trunk/da/d6e/tutorial_py_geometric_transformations.html
26. Towards Data Science: An Intuitive Explanation of Connectionist Temporal Classification [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c
27. Medium: MVC for Flask Application [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@shravan007.c/mvc-for-flask-application-a636e6f58d72

28. Machine Learning Mastery: A Gentle Introduction to Transfer Learning for Deep Learning [Електронний ресурс] – Режим доступу до ресурсу: https://machinelearningmastery.com/transfer-learning-for-deep-learning/
29. AWS: Amazon EC2 [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ru/ec2/.
30. Google Cloud: Google Cloud Products [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/gcp//

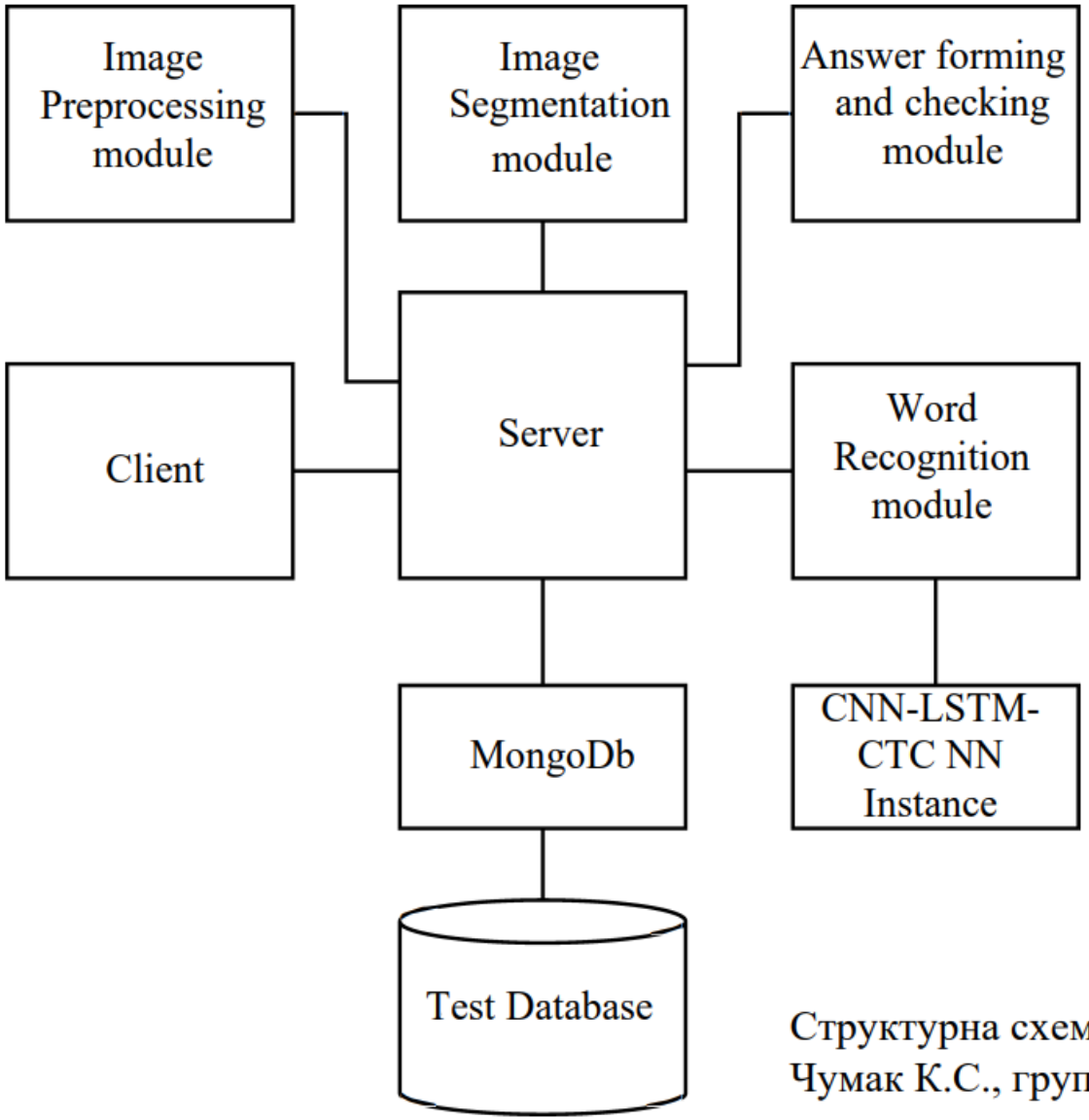
ДОДАТКИ

Додаток 1
Копії графічних матеріалів

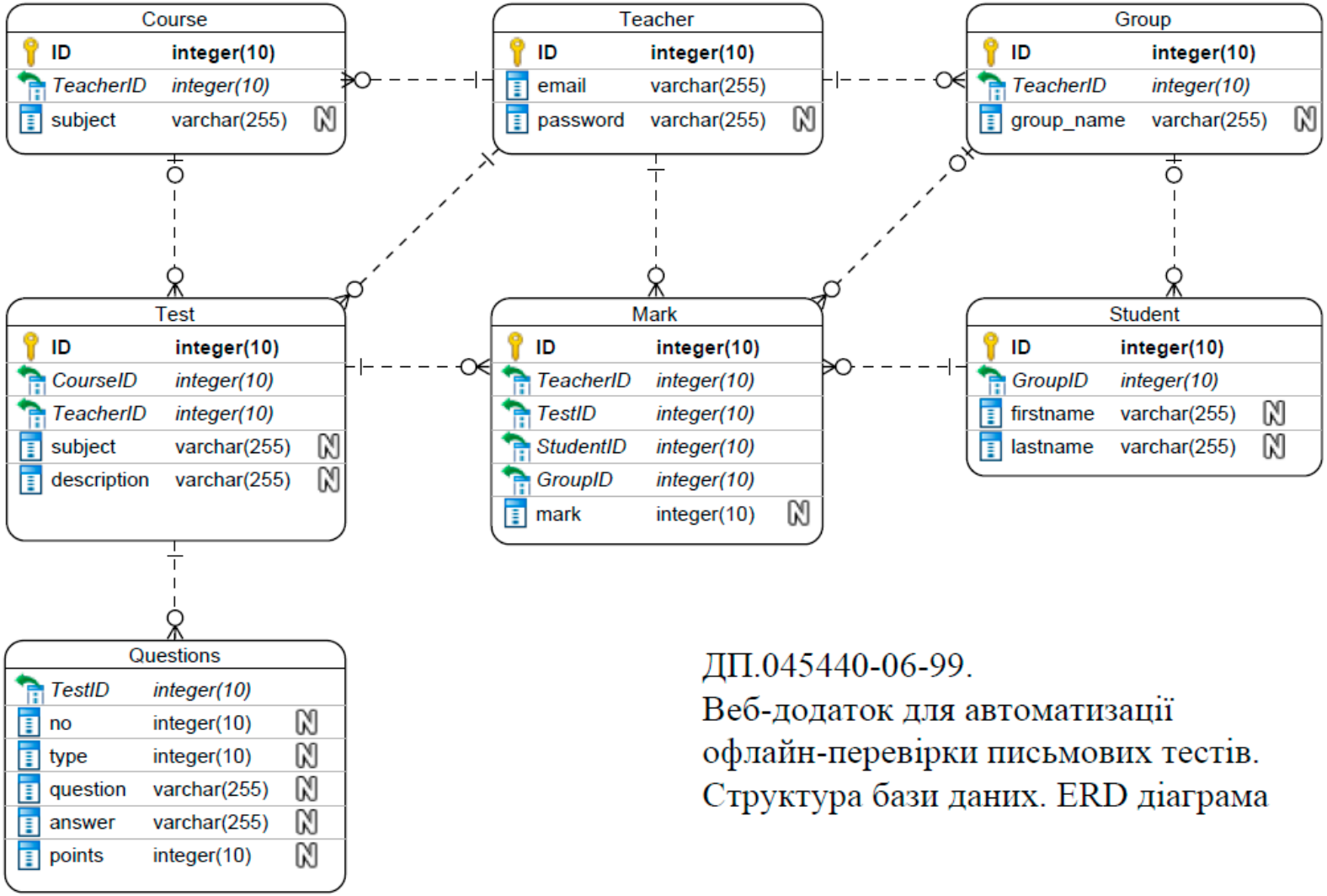
Архітектура нейронної мережі



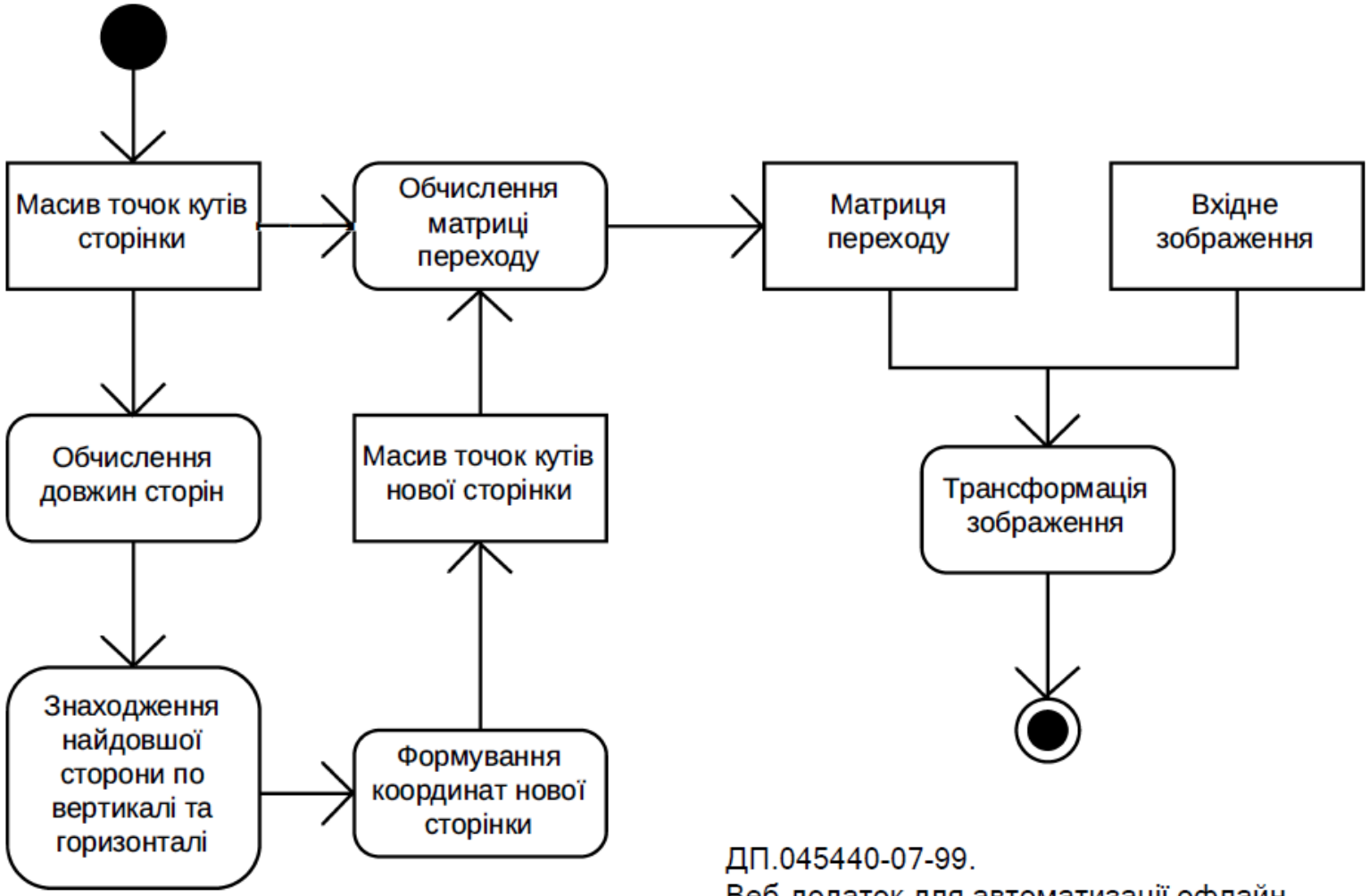
Чумак К.С., група КП-51



Структурна схема системи.
Чумак К.С., група КП-51



ДП.045440-06-99.
Веб-додаток для автоматизації офлайн-перевірки письмових тестів.
Структура бази даних. ERD діаграма



ДП.045440-07-99.
Веб-додаток для автоматизації офлайн-перевірки письмових тестів. Алгоритм трансформування по чотирьох точках. Діаграма діяльності

Додаток 2
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО"

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

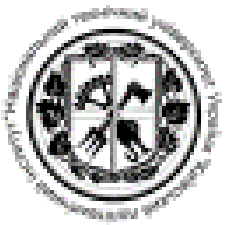
ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ОФЛАЙН-ПЕРЕВІРКИ ПИСЬМОВИХ ТЕСТІВ

Виконала: К.С. Чумак

Науковий керівник: ст. викл. Р.А. Гадиняк

Київ – 2019

АКТУАЛЬНІСТЬ



- Тестування необхідне для моніторингу знань.
- Перехід до повної автоматизації навчання.
- Офлайн-тестування найбільш доступне та репрезентативне.
- Перевірка паперових тестів – монотонна задача.

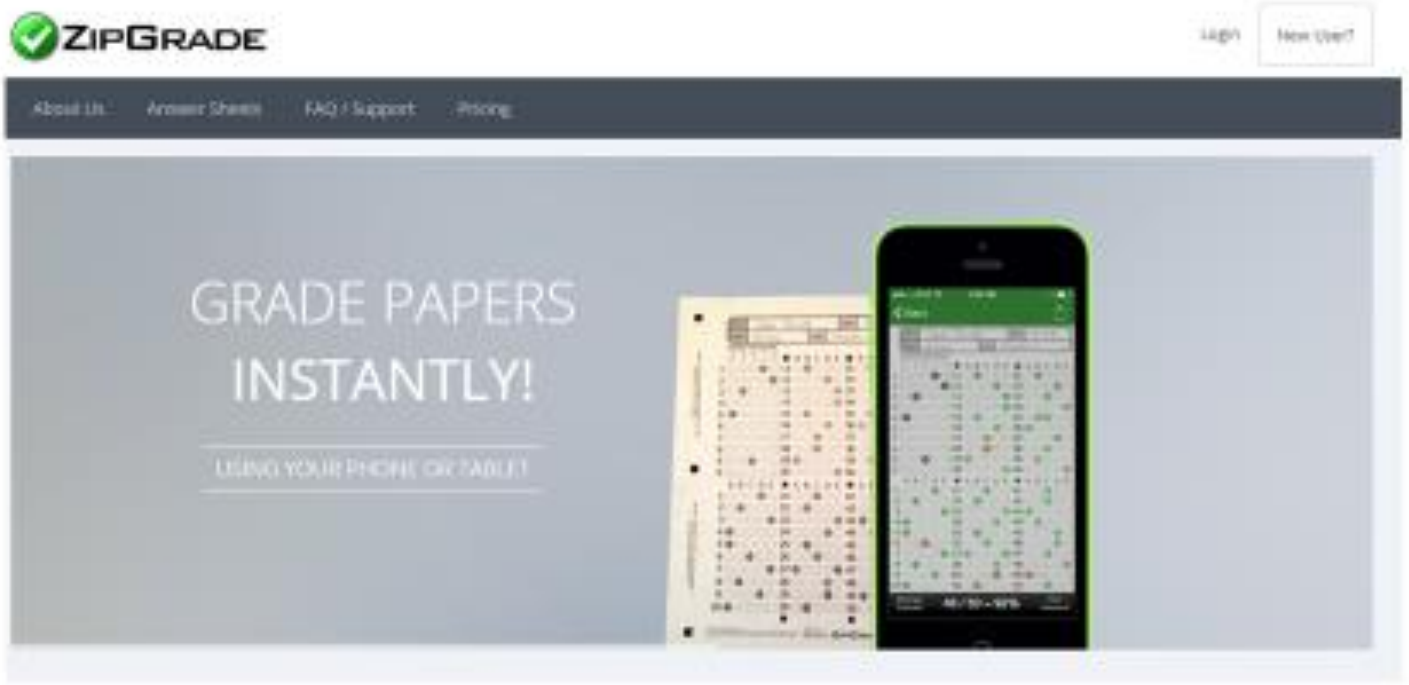
ПОСТАНОВКА ЗАДАЧІ

Мета проекту: розробити веб-додаток для автоматизації процесу перевірки письмових тестів.

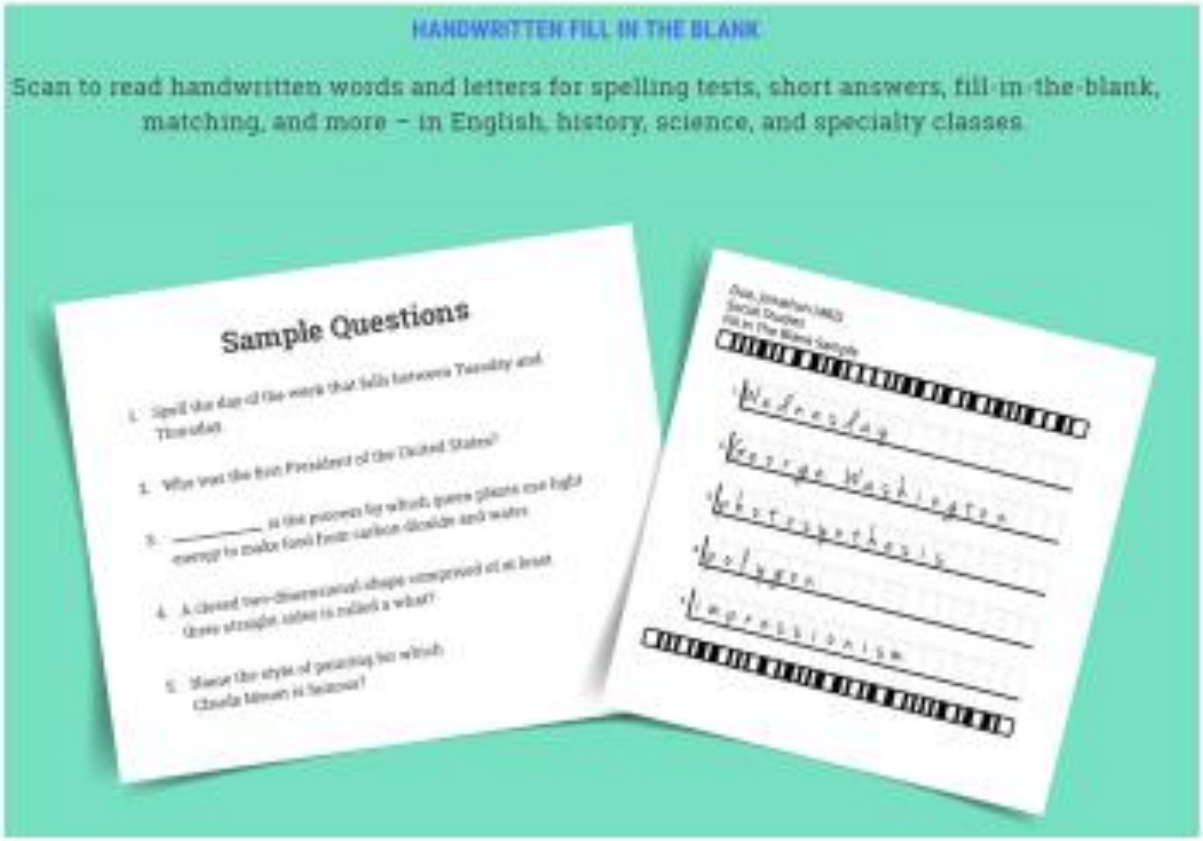
Завдання:

1. Проаналізувати методи розпізнавання та сегментації рукописного тексту.
2. Розробити модель розпізнавання тексту.
3. Розробити веб-інтерфейс.
4. Протестувати роботу системи.

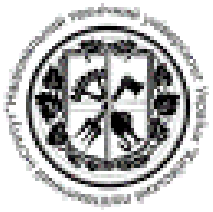
ІСНУЮЧІ АНАЛОГИ. ZIPGRADE.



ІСНУЮЧІ АНАЛОГИ. GRADECAM.



АРХІТЕКТУРА СИСТЕМИ



- Клієнт-серверна взаємодія
- База даних
- Модуль обробки зображення
- Модуль сегментації тексту
- Модуль розпізнавання слів
- Модуль формування та перевірки відповіді
- Нейронна мережа

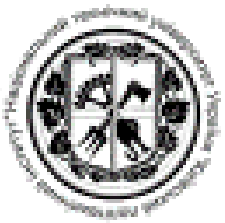
ЗАСОБИ РОЗРОБЛЕННЯ

Bootstrap

РОЗРОБЛЕНІ ЗАСОБИ. СТРУКТУРА БАЗИ ДАНИХ

- Teacher
- Course
- Test
- Questions
- Group
- Student
- Mark

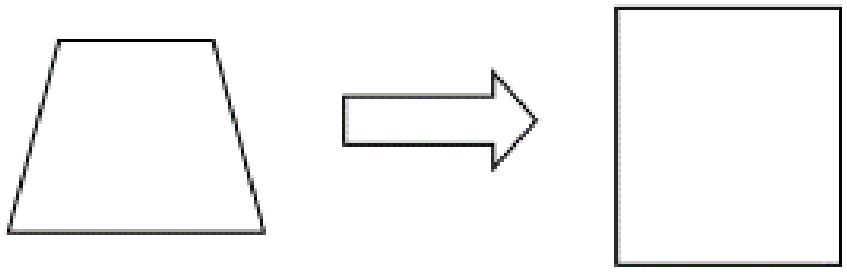
РОЗРОБЛЕНІ АЛГОРИТМИ



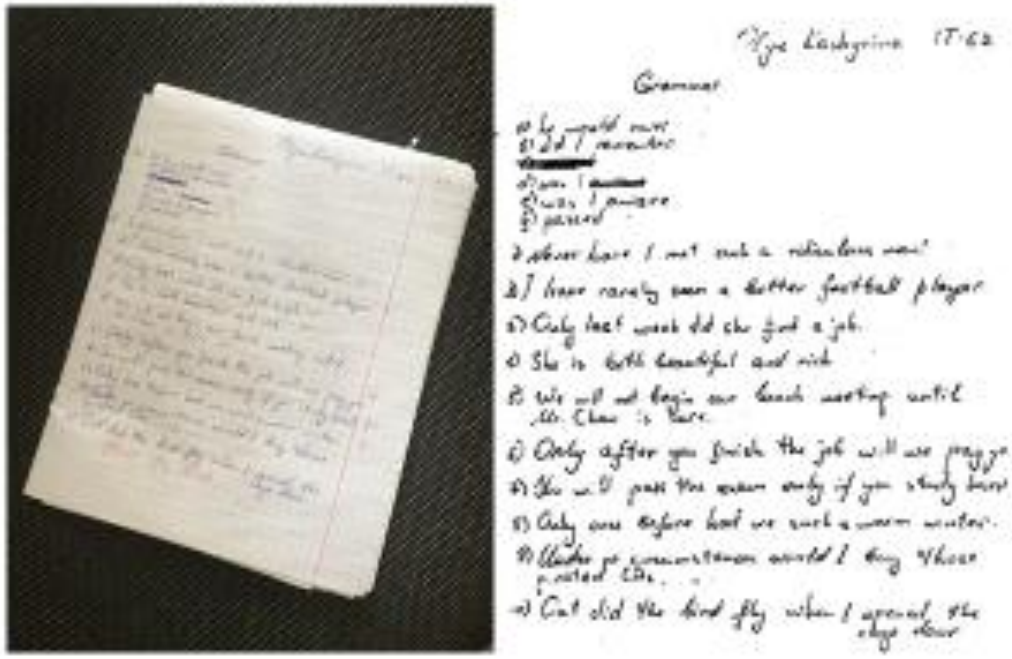
- Алгоритм оброблення вхідного зображення.
- Алгоритм трансформації по чотирьох точках.
- Алгоритм розпізнавання слів на зображенні (Нейронна мережа).
- Алгоритм формування масиву відповідей та їх перевірки.

РОЗРОБЛЕНІ АЛГОРИТМИ. ТРАНСФОРМАЦІЯ ПО ЧОТИРЬОХ ТОЧКАХ

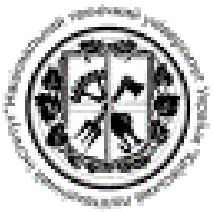
- Знайдено координати кутів сторінки, знаходяться довжини сторін.
- Формується координати нової сторінки.
- Знаходиться матриця переходу.
- Застосовується перспективна трансформація.



РЕЗУЛЬТАТ ОБРОБЛЕННЯ ЗОБРАЖЕННЯ



РОЗРОБЛЕНІ АЛГОРИТМИ. НЕЙРОННА МЕРЕЖА



- 5 згорткових шарів (CNN): фільтри 5x5 та 2x2
- RELU активація
- MaxPool агрегація
- 2 рекурентних шара (LSTM)
- CTC класифікація
- RMSProp оптимізатор

КРИТЕРІЇ ОЦІНЮВАННЯ ЯКОСТІ РОЗРОБЛЕНОЇ МОДЕЛІ

$$CER = \frac{\text{Кількість неправильно визначених символів}}{\text{Кількість символів}}$$

$$WER = \frac{\text{Кількість неправильно визначених слів}}{\text{Кількість слів}}$$

ПОРІВНЯННЯ З ІСНУЮЧИМИ МОДЕЛЯМИ

	CER	WER
1	30.6	-
2	8.3	27.5
3	6.2	20.2
Розроблена модель	10.63	29.4

1. N.Y.Hammerla. Towards Feature Learning for HMM-based Offline Handwriting Recognition / N.Y.Hammerla, T.Ploetz, G.A. Fink, S.Vajda, 2010.
2. J.Puigcerver. Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition?, 2017.
3. A.Chowdhury. An Efficient End-to-End Neural Model for Handwritten Text Recognition/ A.Chowdhury, L.Vig, 2018.

КРИТЕРІЇ ОЦІНЮВАННЯ ЯКОСТІ СИСТЕМИ

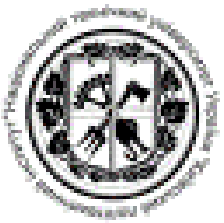
1. Гнучкість у створенні тестів різних типів.
2. Розпізнавання тексту у довільній формі.
3. Побудова графіків.
4. Відсутність необхідності друкування форм.

ПОРІВНЯННЯ З АНАЛОГАМИ

Критерій	ZipGrade	GradeCam	Розроблене ПЗ
1	-	+	+
2	-	-	+
3	-	+	+
4	-	-	+

ПРИКЛАД РОБОТИ ПРОГРАМИ

ВИСНОВКИ



1. Проаналізовано методи розпізнавання рукописного тексту.
2. Розроблено клієнт-серверну архітектуру ПЗ.
3. Розроблено алгоритми обробки зображень та модель розпізнавання тексту.
4. Розроблено структуру бази даних.
5. Розроблено веб-інтерфейс для роботи з системою.
6. Протестовано програмні засоби.

Дякую за увагу!

Додаток 3
Лістинг модуля роботи з нейронною мережею

Model.py

```
from __future__ import division
from __future__ import print_function

import sys
import numpy as np
import tensorflow as tf
import os


class DecoderType:
	BestPath = 0
	BeamSearch = 1
	WordBeamSearch = 2


class Model:
	"minimalistic TF model for HTR"

	# model constants
	batchSize = 50
	imgSize = (128, 32)
	maxTextLen = 32

	def __init__(self, charList, decoderType=DecoderType.BestPath,
mustRestore=False, dump=False):
		"init model: add CNN, RNN and CTC and initialize TF"
		self.dump = dump
		self.charList = charList
		self.decoderType = decoderType
		self.mustRestore = mustRestore
		self.snapID = 0

		# Whether to use normalization over a batch or a population
		self.is_train = tf.placeholder(tf.bool, name='is_train')

		# input image batch
		self.inputImgs = tf.placeholder(tf.float32, shape=(None,
Model.imgSize[0], Model.imgSize[1]))

		# setup CNN, RNN and CTC
		self.setupCNN()
		self.setupRNN()
		self.setupCTC()

		# setup optimizer to train NN
		self.batchesTrained = 0
		self.learningRate = tf.placeholder(tf.float32, shape=[])
		self.update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
		with tf.control_dependencies(self.update_ops):
			self.optimizer =
tf.train.RMSPropOptimizer(self.learningRate).minimize(self.loss)

		# initialize TF
		(self.sess, self.saver) = self.setupTF()


	def setupCNN(self):
		"create CNN layers and return output of these layers"
		cnnIn4d = tf.expand_dims(input=self.inputImgs, axis=3)

		# list of parameters for the layers
		kernelVals = [5, 5, 3, 3, 3]
		featureVals = [1, 32, 64, 128, 128, 256]
```

```
            strideVals = poolVals = [(2,2), (2,2), (1,2), (1,2), (1,2)]
            numLayers = len(strideVals)

            # create layers
            pool = cnnIn4d # input to first CNN layer
            for i in range(numLayers):
                  kernel = tf.Variable(tf.truncated_normal([kernelVals[i],
kernelVals[i], featureVals[i], featureVals[i + 1]], stddev=0.1))
                  conv = tf.nn.conv2d(pool, kernel, padding='SAME',
strides=(1,1,1,1))
                  conv_norm = tf.layers.batch_normalization(conv,
training=self.is_train)
                  relu = tf.nn.relu(conv_norm)
                  pool = tf.nn.max_pool(relu, (1, poolVals[i][0],
poolVals[i][1], 1), (1, strideVals[i][0], strideVals[i][1], 1), 'VALID')

            self.cnnOut4d = pool


      def setupRNN(self):
            "create RNN layers and return output of these layers"
            rnnIn3d = tf.squeeze(self.cnnOut4d, axis=[2])

            # basic cells which is used to build RNN
            numHidden = 256
            cells = [tf.contrib.rnn.LSTMCell(num_units=numHidden,
state_is_tuple=True) for _ in range(2)] # 2 layers

            # stack basic cells
            stacked = tf.contrib.rnn.MultiRNNCell(cells,
state_is_tuple=True)

            # bidirectional RNN
            # BxTxF -> BxTx2H
            ((fw, bw), _) =
tf.nn.bidirectional_dynamic_rnn(cell_fw=stacked, cell_bw=stacked,
inputs=rnnIn3d, dtype=rnnIn3d.dtype)

            # BxTxH + BxTxH -> BxTx2H -> BxTx1X2H
            concat = tf.expand_dims(tf.concat([fw, bw], 2), 2)

            # project output to chars (including blank): BxTx1x2H ->
BxTx1xC -> BxTxC
            kernel = tf.Variable(tf.truncated_normal([1, 1, numHidden * 2,
len(self.charList) + 1], stddev=0.1))
            self.rnnOut3d = tf.squeeze(tf.nn.atrous_conv2d(value=concat,
filters=kernel, rate=1, padding='SAME'), axis=[2])


      def setupCTC(self):
            "create CTC loss and decoder and return them"
            # BxTxC -> TxBxC
            self.ctcIn3dTBC = tf.transpose(self.rnnOut3d, [1, 0, 2])
            # ground truth text as sparse tensor
            self.gtTexts = tf.SparseTensor(tf.placeholder(tf.int64,
shape=[None, 2]) , tf.placeholder(tf.int32, [None]),
tf.placeholder(tf.int64, [2]))

            # calc loss for batch
            self.seqLen = tf.placeholder(tf.int32, [None])
            self.loss = tf.reduce_mean(tf.nn.ctc_loss(labels=self.gtTexts,
inputs=self.ctcIn3dTBC, sequence_length=self.seqLen,
ctc_merge_repeated=True))
```

```
            # calc loss for each element to compute label probability
            self.savedCtcInput = tf.placeholder(tf.float32,
shape=[Model.maxTextLen, None, len(self.charList) + 1])
            self.lossPerElement = tf.nn.ctc_loss(labels=self.gtTexts,
inputs=self.savedCtcInput, sequence_length=self.seqLen,
ctc_merge_repeated=True)

            # decoder: either best path decoding or beam search decoding
            if self.decoderType == DecoderType.BestPath:
                  self.decoder =
tf.nn.ctc_greedy_decoder(inputs=self.ctcIn3dTBC,
sequence_length=self.seqLen)
            elif self.decoderType == DecoderType.BeamSearch:
                  self.decoder =
tf.nn.ctc_beam_search_decoder(inputs=self.ctcIn3dTBC,
sequence_length=self.seqLen, beam_width=50, merge_repeated=False)
            elif self.decoderType == DecoderType.WordBeamSearch:
                  # import compiled word beam search operation (see
https://github.com/githubharald/CTCWordBeamSearch)
                  word_beam_search_module =
tf.load_op_library('TFWordBeamSearch.so')

                  # prepare information about language (dictionary,
characters in dataset, characters forming words)
                  chars = str().join(self.charList)
                  wordChars =
open('../model/wordCharList.txt').read().splitlines()[0]
                  corpus = open('../data/corpus.txt').read()

                  # decode using the "Words" mode of word beam search
                  self.decoder =
word_beam_search_module.word_beam_search(tf.nn.softmax(self.ctcIn3dTBC,
dim=2), 50, 'Words', 0.0, corpus.encode('utf8'), chars.encode('utf8'),
wordChars.encode('utf8'))


      def setupTF(self):
            "initialize TF"
            print('Python: '+sys.version)
            print('Tensorflow: '+tf.__version__)

            sess=tf.Session() # TF session

            saver = tf.train.Saver(max_to_keep=1) # saver saves model to
file
            modelDir = '../model/'
            latestSnapshot = tf.train.latest_checkpoint(modelDir) # is
there a saved model?

            # if model must be restored (for inference), there must be a
snapshot
            if self.mustRestore and not latestSnapshot:
                  raise Exception('No saved model found in: ' + modelDir)

            # load saved model if available
            if latestSnapshot:
                  print('Init with stored values from ' + latestSnapshot)
                  saver.restore(sess, latestSnapshot)
            else:
                  print('Init with new values')
                  sess.run(tf.global_variables_initializer())

            return (sess,saver)
```

```
	def toSparse(self, texts):
		"put ground truth texts into sparse tensor for ctc_loss"
		indices = []
		values = []
		shape = [len(texts), 0] # last entry must be max(labelList[i])

		# go over all texts
		for (batchElement, text) in enumerate(texts):
			# convert to string of label (i.e. class-ids)
			labelStr = [self.charList.index(c) for c in text]
			# sparse tensor must have size of max. label-string
			if len(labelStr) > shape[1]:
				shape[1] = len(labelStr)
			# put each label into sparse tensor
			for (i, label) in enumerate(labelStr):
				indices.append([batchElement, i])
				values.append(label)

		return (indices, values, shape)


	def decoderOutputToText(self, ctcOutput, batchSize):
		"extract texts from output of CTC decoder"

		# contains string of labels for each batch element
		encodedLabelStrs = [[] for i in range(batchSize)]

		# word beam search: label strings terminated by blank
		if self.decoderType == DecoderType.WordBeamSearch:
			blank=len(self.charList)
			for b in range(batchSize):
				for label in ctcOutput[b]:
					if label==blank:
						break
					encodedLabelStrs[b].append(label)

		# TF decoders: label strings are contained in sparse tensor
		else:
			# ctc returns tuple, first element is SparseTensor
			decoded=ctcOutput[0][0]

			# go over all indices and save mapping: batch -> values
			idxDict = { b : [] for b in range(batchSize) }
			for (idx, idx2d) in enumerate(decoded.indices):
				label = decoded.values[idx]
				batchElement = idx2d[0] # index according to [b,t]
				encodedLabelStrs[batchElement].append(label)

		# map labels to chars for all batch elements
		return [str().join([self.charList[c] for c in labelStr]) for
labelStr in encodedLabelStrs]


	def trainBatch(self, batch):
		"feed a batch into the NN to train it"
		numBatchElements = len(batch.imgs)
		sparse = self.toSparse(batch.gtTexts)
		rate = 0.01 if self.batchesTrained < 10 else (0.001 if
self.batchesTrained < 10000 else 0.0001) # decay learning rate
		evalList = [self.optimizer, self.loss]
		feedDict = {self.inputImgs : batch.imgs, self.gtTexts : sparse
, self.seqLen : [Model.maxTextLen] * numBatchElements, self.learningRate :
rate, self.is_train: True}
```

```
            (_, lossVal) = self.sess.run(evalList, feedDict)
            self.batchesTrained += 1
            return lossVal


      def dumpNNOutput(self, rnnOutput):
            "dump the output of the NN to CSV file(s)"
            dumpDir = '../dump/'
            if not os.path.isdir(dumpDir):
                  os.mkdir(dumpDir)

            # iterate over all batch elements and create a CSV file for
each one
            maxT, maxB, maxC = rnnOutput.shape
            for b in range(maxB):
                  csv = ''
                  for t in range(maxT):
                        for c in range(maxC):
                              csv += str(rnnOutput[t, b, c]) + ';'
                        csv += '\n'
                  fn = dumpDir + 'rnnOutput_'+str(b)+'.csv'
                  print('Write dump of NN to file: ' + fn)
                  with open(fn, 'w') as f:
                        f.write(csv)


      def inferBatch(self, batch, calcProbability=False,
probabilityOfGT=False):
            "feed a batch into the NN to recognize the texts"

            # decode, optionally save RNN output
            numBatchElements = len(batch.imgs)
            evalRnnOutput = self.dump or calcProbability
            evalList = [self.decoder] + ([self.ctcIn3dTBC] if evalRnnOutput
else [])
            feedDict = {self.inputImgs : batch.imgs, self.seqLen :
[Model.maxTextLen] * numBatchElements, self.is_train: False}
            evalRes = self.sess.run(evalList, feedDict)
            decoded = evalRes[0]
            texts = self.decoderOutputToText(decoded, numBatchElements)

            # feed RNN output and recognized text into CTC loss to compute
labeling probability
            probs = None
            if calcProbability:
                  sparse = self.toSparse(batch.gtTexts) if probabilityOfGT
else self.toSparse(texts)
                  ctcInput = evalRes[1]
                  evalList = self.lossPerElement
                  feedDict = {self.savedCtcInput : ctcInput, self.gtTexts :
sparse, self.seqLen : [Model.maxTextLen] * numBatchElements, self.is_train:
False}
                  lossVals = self.sess.run(evalList, feedDict)
                  probs = np.exp(-lossVals)

            # dump the output of the NN to CSV file(s)
            if self.dump:
                  self.dumpNNOutput(evalRes[1])

            return (texts, probs)


      def save(self):
            "save model to file"
```

```
            self.snapID += 1
            self.saver.save(self.sess, '../model/snapshot',
global_step=self.snapID)
```

analyse.py

```
from __future__ import division
from __future__ import print_function

import sys
import math
import pickle
import copy
import numpy as np
import cv2
import matplotlib.pyplot as plt
from DataLoader import Batch
from Model import Model, DecoderType
from SamplePreprocessor import preprocess


# constants like filepaths
class Constants:
      "filenames and paths to data"
      fnCharList = '../model/charList.txt'
      fnAnalyze = '../data/analyze.png'
      fnPixelRelevance = '../data/pixelRelevance.npy'
      fnTranslationInvariance = '../data/translationInvariance.npy'
      fnTranslationInvarianceTexts =
'../data/translationInvarianceTexts.pickle'
      gtText = 'are'
      distribution = 'histogram' # 'histogram' or 'uniform'


def odds(val):
      return val / (1 - val)


def weightOfEvidence(origProb, margProb):
      return math.log2(odds(origProb)) - math.log2(odds(margProb))


def analyzePixelRelevance():
      "simplified implementation of paper: Zintgraf et al - Visualizing
Deep Neural Network Decisions: Prediction Difference Analysis"

      # setup model
      model = Model(open(Constants.fnCharList).read(),
DecoderType.BestPath, mustRestore=True)

      # read image and specify ground-truth text
      img = cv2.imread(Constants.fnAnalyze, cv2.IMREAD_GRAYSCALE)
      (w, h) = img.shape
      assert Model.imgSize[1] == w

      # compute probability of gt text in original image
      batch = Batch([Constants.gtText], [preprocess(img, Model.imgSize)])
      (_, probs) = model.inferBatch(batch, calcProbability=True,
probabilityOfGT=True)
      origProb = probs[0]

      grayValues = [0, 63, 127, 191, 255]
      if Constants.distribution == 'histogram':
```

```
            bins = [0, 31, 95, 159, 223, 255]
            (hist, _) = np.histogram(img, bins=bins)
            pixelProb = hist / sum(hist)
      elif Constants.distribution == 'uniform':
            pixelProb = [1.0 / len(grayValues) for _ in grayValues]
      else:
            raise Exception('unknown value for Constants.distribution')

      # iterate over all pixels in image
      pixelRelevance = np.zeros(img.shape, np.float32)
      for x in range(w):
            for y in range(h):

                  # try a subset of possible grayvalues of pixel (x,y)
                  imgsMarginalized = []
                  for g in grayValues:
                        imgChanged = copy.deepcopy(img)
                        imgChanged[x, y] = g
                        imgsMarginalized.append(preprocess(imgChanged,
Model.imgSize))

                  # put them all into one batch
                  batch = Batch([Constants.gtText]*len(imgsMarginalized),
imgsMarginalized)

                  # compute probabilities
                  (_, probs) = model.inferBatch(batch,
calcProbability=True, probabilityOfGT=True)

                  # marginalize over pixel value (assume uniform
distribution)
                  margProb = sum([probs[i] * pixelProb[i] for i in
range(len(grayValues))])

                  pixelRelevance[x, y] = weightOfEvidence(origProb,
margProb)

                  print(x, y, pixelRelevance[x, y], origProb, margProb)

      np.save(Constants.fnPixelRelevance, pixelRelevance)


def analyzeTranslationInvariance():
      # setup model
      model = Model(open(Constants.fnCharList).read(),
DecoderType.BestPath, mustRestore=True)

      # read image and specify ground-truth text
      img = cv2.imread(Constants.fnAnalyze, cv2.IMREAD_GRAYSCALE)
      (w, h) = img.shape
      assert Model.imgSize[1] == w

      imgList = []
      for dy in range(Model.imgSize[0]-h+1):
            targetImg = np.ones((Model.imgSize[1], Model.imgSize[0])) * 255
            targetImg[:,dy:h+dy] = img
            imgList.append(preprocess(targetImg, Model.imgSize))

      # put images and gt texts into batch
      batch = Batch([Constants.gtText]*len(imgList), imgList)

      # compute probabilities
      (texts, probs) = model.inferBatch(batch, calcProbability=True,
probabilityOfGT=True)
```

```
	# save results to file
	f = open(Constants.fnTranslationInvarianceTexts, 'wb')
	pickle.dump(texts, f)
	f.close()
	np.save(Constants.fnTranslationInvariance, probs)


def showResults():
	# 1. pixel relevance
	pixelRelevance = np.load(Constants.fnPixelRelevance)
	plt.figure('Pixel relevance')

	plt.imshow(pixelRelevance, cmap=plt.cm.jet, vmin=-0.25, vmax=0.25)
	plt.colorbar()

	img = cv2.imread(Constants.fnAnalyze, cv2.IMREAD_GRAYSCALE)
	plt.imshow(img, cmap=plt.cm.gray, alpha=.4)


	# 2. translation invariance
	probs = np.load(Constants.fnTranslationInvariance)
	f = open(Constants.fnTranslationInvarianceTexts, 'rb')
	texts = pickle.load(f)
	texts = ['%d:'%i + texts[i] for i in range(len(texts))]
	f.close()

	plt.figure('Translation invariance')

	plt.plot(probs, 'o-')
	plt.xticks(np.arange(len(texts)), texts, rotation='vertical')
	plt.xlabel('horizontal translation and best path')
	plt.ylabel('text probability of "%s"'%Constants.gtText)

	# show both plots
	plt.show()


if __name__ == '__main__':
	if len(sys.argv)>1:
		if sys.argv[1]=='--relevance':
			print('Analyze pixel relevance')
			analyzePixelRelevance()
		elif sys.argv[1]=='--invariance':
			print('Analyze translation invariance')
			analyzeTranslationInvariance()
	else:
		print('Show results')
		showResults()
```

preprocessor.py

```
from __future__ import division
from __future__ import print_function

import random
import numpy as np
import cv2


def preprocess(img, imgSize, dataAugmentation=False):
	"put img into target img of size imgSize, transpose for TF and
normalize gray-values"

	# there are damaged files in IAM dataset - just use black image
instead
	if img is None:
		img = np.zeros([imgSize[1], imgSize[0]])

	# increase dataset size by applying random stretches to the images
	if dataAugmentation:
		stretch = (random.random() - 0.5) # -0.5 .. +0.5
		wStretched = max(int(img.shape[1] * (1 + stretch)), 1) # random
width, but at least 1
		img = cv2.resize(img, (wStretched, img.shape[0])) # stretch
horizontally by factor 0.5 .. 1.5

	# create target image and copy sample image into it
	(wt, ht) = imgSize
	(h, w) = img.shape
	fx = w / wt
	fy = h / ht
	f = max(fx, fy)
	newSize = (max(min(wt, int(w / f)), 1), max(min(ht, int(h / f)), 1))
# scale according to f (result at least 1 and at most wt or ht)
	img = cv2.resize(img, newSize)
	target = np.ones([ht, wt]) * 255
	target[0:newSize[1], 0:newSize[0]] = img

	# transpose for TF
	img = cv2.transpose(target)

	# normalize
	(m, s) = cv2.meanStdDev(img)
	m = m[0][0]
	s = s[0][0]
	img = img - m
	img = img / s if s>0 else img
	return img
```

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" __ _____ 2018 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ОФЛАЙН-ПЕРЕВІРКИ ПИСЬМОВИХ ТЕСТІВ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А. Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

__ ____ К.С. Чумак

2018

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Веб-додаток для автоматизації офлайн-перевірки письмових тестів, написаний на мові програмування Python з використанням фреймворку Flask.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1. Функціональна працездатність елементів сторінок web-додатку.
2. Наявність доступу до бази даних тестів.
3. Розпізнавання рукописного тексту з зображення.
4. Взаємодію сервера з модулем розпізнавання рукописного тексту з зображення.
5. Формування масиву відповідей та виставлення рекомендованої оцінки.
6. Забезпечення коректної обробки запитів від користувача.
7. Забезпечення належного рівня безпеки даних.
8. Зручність роботи з web-додатком.
9. Відповідність дизайну вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1. Функціональне тестування, зокрема на рівні Critical path test (базове тестування).

2. Тестування продуктивності програмного забезпечення, зокрема Performance testing (тестування стабільності).
3. Тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність web-додатку перевіряється шляхом:

1. Динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати.
2. Динамічного ручного тестування на відповідність функціональним вимогам.
3. Статичного тестування коду.
4. Тестування web-ресурсу в різних web-браузерах.
5. Тестування при максимальному навантаженні.
6. Тестування стабільності роботи при різних умовах.
7. Тестування зручності використання.
8. Тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ОФЛАЙН-ПЕРЕВІРКИ ПИСЬМОВИХ ТЕСТІВ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____ Р.А. Гадиняк

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ К.С. Чумак

2019

ЗМІСТ

1. Опис структури програмних засобів

Програмні засоби включають в себе web-додаток, що складається із статичних web-сторінок та web-сторінок, вміст яких формується динамічно. Web-додаток є одномовним з англійською мовою.

До статичних належать наступні web-сторінки:

- сторінка реєстрації;
- сторінка авторизації .

Динамічна частина web-додатку включає наступні web-сторінки:

- списки груп;
- сторінка створення тесту;
- створені курси;
- створені тести;
- сторінка перегляду графіків;
- сторінка перегляду тесту;
- сторінка перевірки тесту;

Кожна web-сторінка містить дві навігаційні панелі. Згори сторінки розташована панель авторизації, яка містить кнопку виходу з системи для авторизованих користувачів та кнопки реєстрації та авторизації – для неавторизованих. Ліворуч на сторінці у авторизованих користувачів розташовується панель навігації у системі, яка доступна з усіх інших сторінок.

2. Опис вмісту статичних web-сторінок

Оскільки усі функції системи доступні лише зареєстрованим користувачам, неавторизований користувач має доступ лише до інформаційної сторінки «Про систему» та сторінок реєстрації та авторизації.

Сторінка авторизації (рис. 1) містить поля для вводу електронної пошти та паролю користувача, кнопку «Запам’ятати мене», кнопку ініціації

входу в систему, а також посилання на форму реєстрації для нових користувачів.

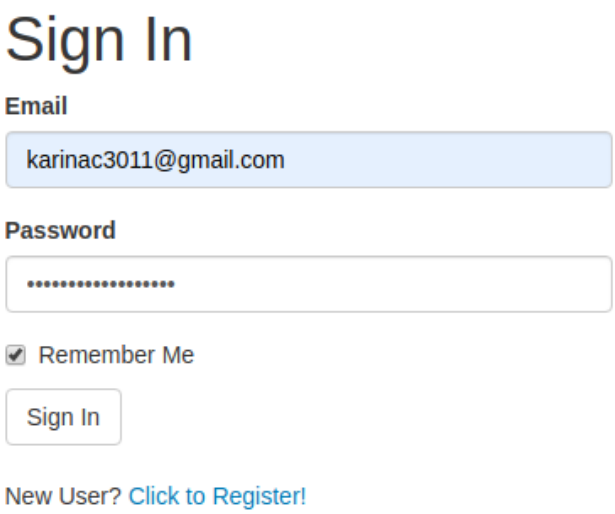


Рис. 1. Сторінка авторизації

Сторінка реєстрації (рис. 2) містить поля для вводу електронної пошти, паролю користувача, поле для підтвердження паролю та кнопку ініціації реєстрації у системі. В обох формах введений пароль прихований за допомогою спеціальних символів у формі кружків.

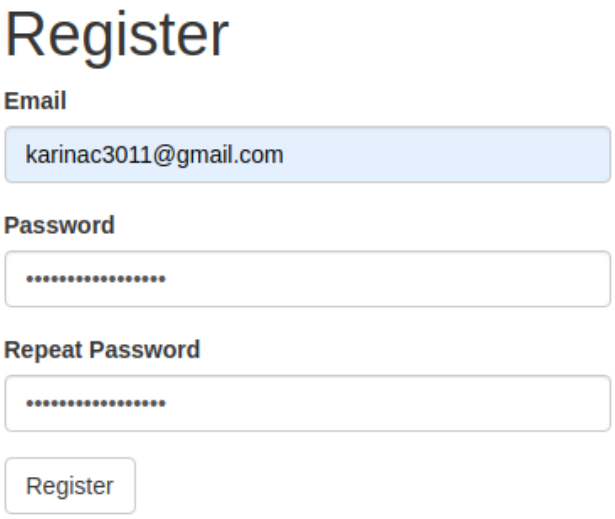


Рис. 2. Сторінка реєстрації

3. Процедура реєстрації користувача

Реєстрація користувача відбувається на сторінці реєстрації. Користувач має ввести свою електронну пошту та пароль. Для уникнення можливих помилок у паролі необхідно ввести його ще раз у поле «Repeat password». Після цього треба натиснути кнопку «Register», що призведе до реєстрації користувача у системі та автоматичної авторизації.

При спробі зареєструватися під електронною поштою, яка вже існує у системі, користувач побачить помилку реєстрації з проханням обрати іншу поштову скриньку.

4. Процедура авторизації користувача

Авторизація користувача відбувається на сторінці авторизації. Користувач має ввести свою електронну пошту та пароль. Після цього треба натиснути кнопку «Sign in», що призведе до авторизації користувача у системі. За бажанням, користувач може обрати опцію «Remember me», що залишить його авторизованим у системі, поки він не вийде з неї явно.

При неправильно введеній пошті або паролю, користувач не зможе авторизуватися у системі, та побачить повідомлення про помилку.

5. Перегляд створених тестів

Для перегляду тестів, що були створені поточним користувачем, необхідно обрати у лівому навігаційному меню пункт «Tests», що відкриє сторінку курсів (рис. 3). Кожен елемент зі списку курсів є активним посиланням на сторінку тестів цього курсу. Якщо при створенні тесту користувач не вказав курс, до якого він належить, такий тест можна знайти у категорії «Other tests».

Рис. 3. Сторінка перегляду курсів

Обравши в наведеному списку курс, користувач потрапляє на сторінку перегляду тестів (рис. 4) з обраного курсу. Кожен тест у списку містить інформацію про дату його створення, назву курсу, короткий опис тесту та є посиланням на сторінку перегляду обраного тесту.

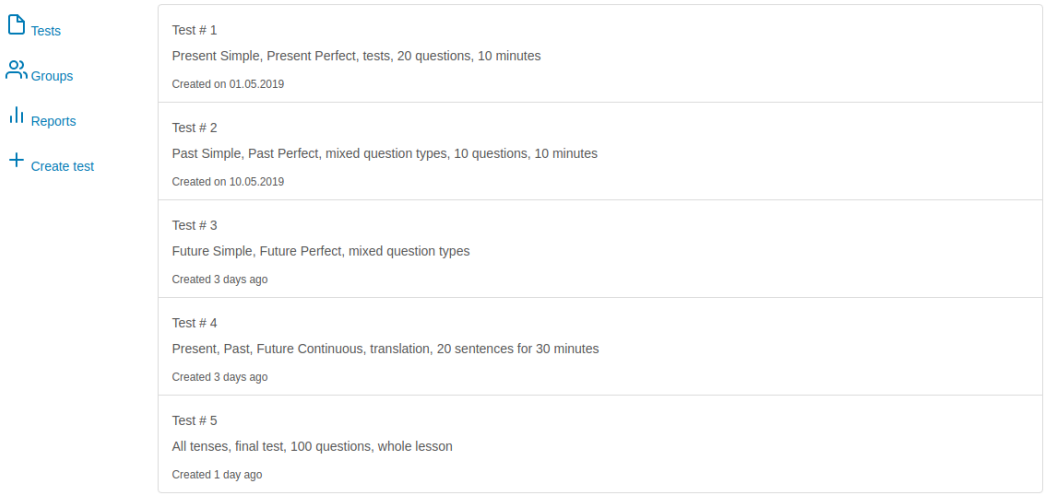


Рис. 4. Сторінка перегляду тестів

6. Перегляд обраного тесту

На сторінці перегляду тестів необхідно обрати тест на натиснути на нього. Після цього відкриється сторінка перегляду тесту (рис. 5).

На цій сторінці питання тесту подано у вигляді таблиці. Над таблицею міститься предмет, до якого належить цей тест, короткий опис тесту а також кнопка «Start checking», яка призведе до сторінки перевірки тесту.

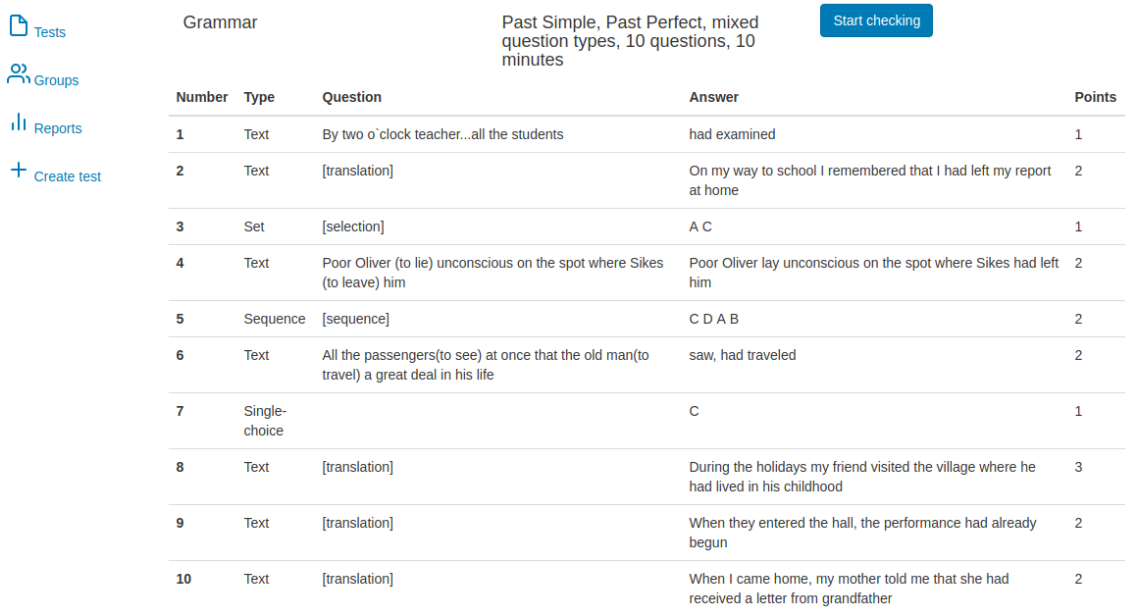


Рис. 5. Сторінка перегляду тесту

7. Перевірка тесту

Для початку перевірки тесту необхідно натиснути кнопку «Start checking» на сторінці перегляду тесту. Після цього відкриється сторінка перевірки (рис. 6), на якій міститься кнопка завантаження зображення, кнопка перевірки роботи та форма виставлення оцінки.

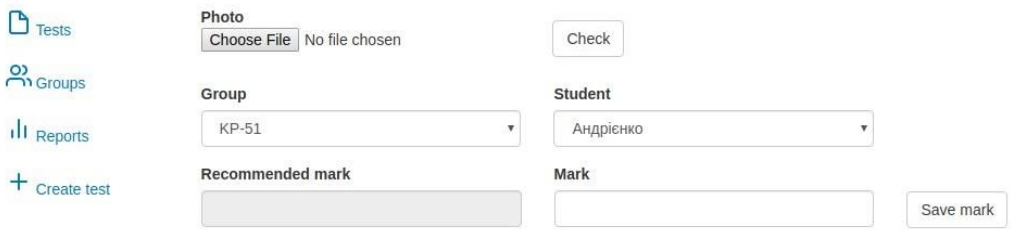


Рис. 6. Сторінка перевірки тесту до початку перевірки

Для перевірки роботи необхідно обрати файл, натиснувши на кнопку «Choose» та відправити його на перевірку, натиснувши на кнопку «Check». Після цього під формою виставлення оцінки з’явиться оброблене вхідне

зображення, колонка з розпізнаними відповідями студента та колонка з правильними відповідями (рис. 7).

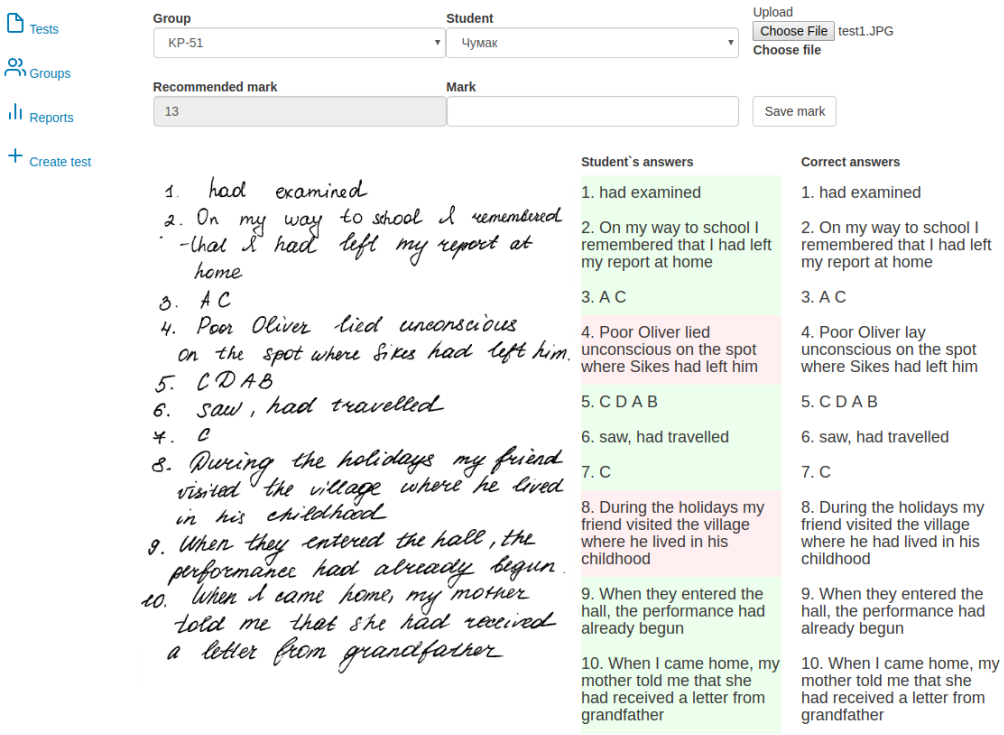


Рис. 7. Сторінка перевірки тесту з перевіреною роботою

У формі виставлення оцінки в полі «Recommended mark» з’явиться число. Студентські відповіді, які визначені як правильні, підсвічуються зеленим кольором, а неправильні – червоним.

Для виставлення оцінки необхідно перевірити неправильні відповіді, записати остаточну оцінку в поле «Mark», обрати групу та прізвище студента зі списку та натиснути кнопку «Save mark».

8. Створення нової групи студентів

Для створення нової групи необхідно обрати у лівому навігаційному меню пункт «Groups». Після цього відкриється сторінка перегляду груп, яка містить форму створення групи та список вже створених груп (рис. 8).

У поле «Group Number» необхідно ввести номер групи, після цього натиснути на кнопу «Choose File», обрати необхідний CSV файл, та натиснути на кнопку «Import». Після цього обрана група буде створена та додана до списку під формою.

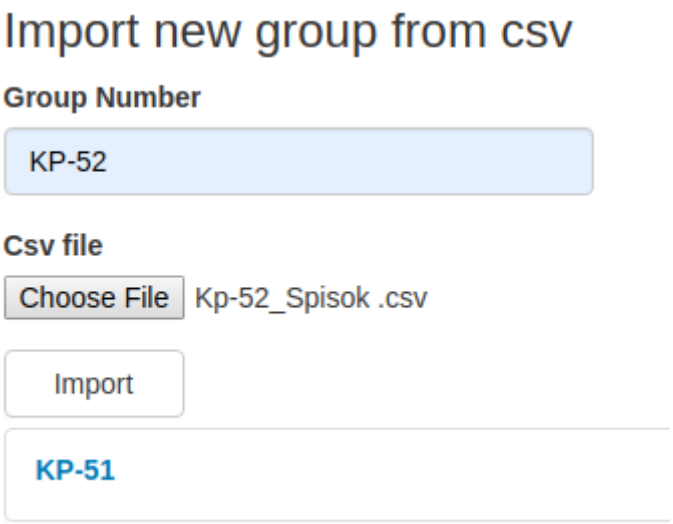


Рис. 8. Сторінка створення групи

9. Створення тесту

Для створення нового необхідно обрати у лівому навігаційному меню пункт «New test». Після цього відкриється сторінка створення тесту (рис. 9). Перед початком створення тестів вона має поле «Subject», поле «Description», а також форму додавання питання.

Спочатку необхідно ввести назву курсу в поле «Subject», короткий опис тесту в поле «Description» та розпочати процес додавання питань. Для цього для кожного питання необхідно обрати його тип з випадаючого списку «Type», ввести текст питання у поле «Question» (необов'язково), ввести відповідь у поле «Answer» та кількість балів у поле «Points» та натиснути на кнопку «Add question». Після цього питання додається до тесту та показується у таблиці під формою додавання питання (рис. 10). Ці

кроки необхідно повторити для кожного питання тесту та натиснути кнопку «Save test» для того, щоб зберегти тест.

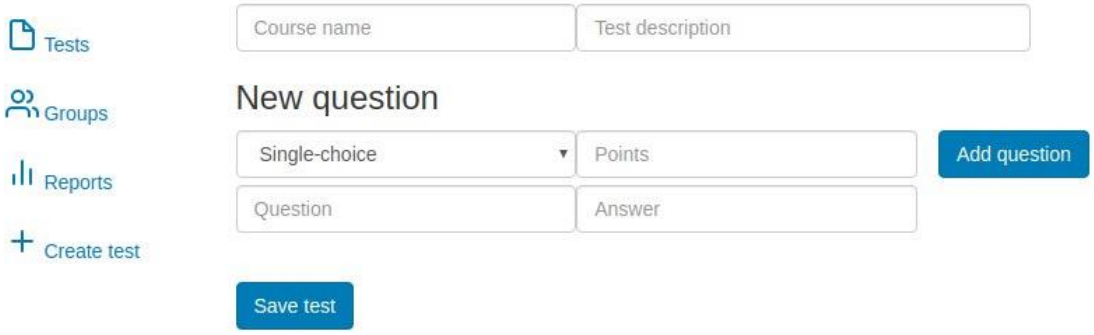


Рис. 9. Сторінка створення тесту

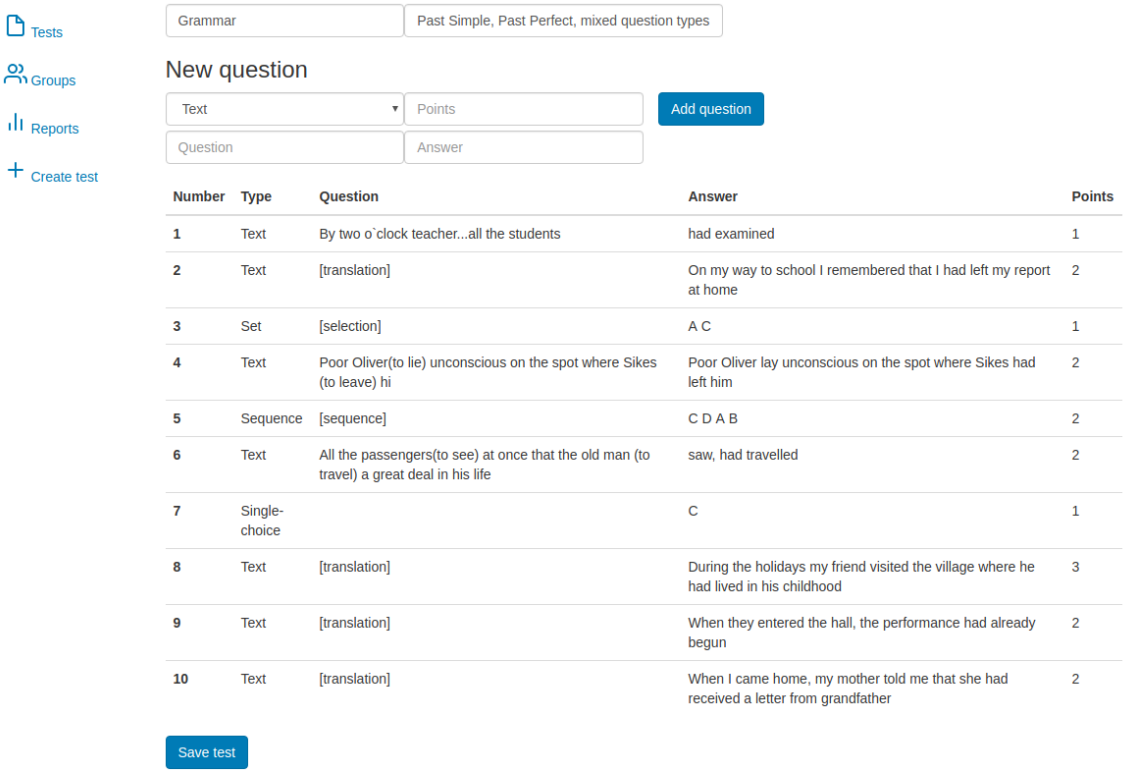


Рис. 10. Сторінка створення тесту після додавання питань

10. Процедура перегляду графіків

Для перегляду графіків необхідно обрати у лівому навігаційному меню пункт «Reports». Після цього відкриється сторінка взаємодії з

графіками (рис. 11). Вона містить три випадаючих списки: «Plot type», «Group» та «Test».

Для побудови гістограми необхідно обрати зі списку «Plot type» пункт «histogram». Для побудови графіку типу «boxplot» необхідно обрати відповідну опцію зі списку.

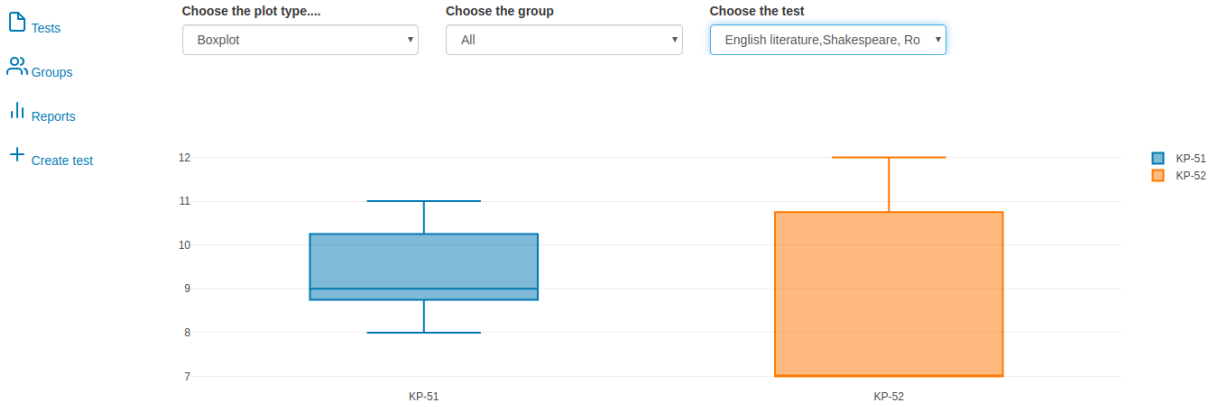


Рис. 11. Сторінка побудови графіків

За замовчуванням графіки будуються для усіх груп з усіх тестів. Це можна змінити, обравши необхідну групу зі списку «Group» та тест зі списку «Test». Графіки змінюються динамічно при зміні параметрів.

Для порівняння результатів груп з одного тесту необхідно вказати тип графіка «boxplot», обрати тест у списку «Test» та обрати опцію «All» у списку груп.